Improved Progressive BKZ with Lattice Sieving and a Two-Step Mode for Solving uSVP

Wenwen Xia[1,⋆], Leizhang Wang[1,⋆], Geng Wang[2], Dawu Gu[1,2], and Baocang Wang[1]

[1] Xidian University
{xiawenwen, lzwang_2}@stu.xidian.edu.cn
bcwang@xidian.edu.cn
[2] Shanghai Jiao Tong University
{wanggxx, dwgu}@sjtu.edu.cn

Abstract. The *unique Shortest Vector Problem* (uSVP) is one of the core hard problems in lattice-based cryptography. In NIST PQC standardization (Kyber, Dilithium), leaky-LWE-Estimator is used to estimate the hardness of LWE-based cryptosystems by reducing LWE to uSVP and considers the primal attack using *Progressive BKZ* (ProBKZ). ProBKZ trivially increases blocksize β and lifts the shortest vector in the final BKZ block to find the unique shortest vector in the full lattice.

In this paper, we show that a ProBKZ algorithm as above (we call it a BKZ-only mode) is not the best way to solve uSVP. So we present a two-step mode to solve it, where the ProBKZ algorithm is followed by a sieving algorithm with the dimension larger than the blocksize of BKZ. While instantiating our two-step mode with the sieving algorithm `Pump` and *Pump-and-jump BKZ* (PnjBKZ) presented in G6K, which are the state-of-art sieving and BKZ implementations, we show that our algorithm is not only better than the BKZ-only mode but also better than the heuristic uSVP solving algorithm in G6K.

However, a ProBKZ with the heuristic parameter selection in leaky-LWE-Estimator or the optimized parameter selection in the literature (Yoshinori Aono *et al.* at Asiacrypt 2016), is insufficient in optimizing the efficiency of a two-step solving algorithm. To find the best parameters, we design a PnjBKZ simulator which allows the choice of value `jump` to be more than 1. Based on the newly designed simulator, we give a blocksize and `jump` strategy selection algorithm, which can achieve the best simulated efficiency in solving uSVP instances. Combining all the things above, we get a new lattice solving algorithm called *Improved Progressive PnjBKZ* (ProPnjBKZ for short).

We test the efficiency of our ProPnjBKZ with the TU Darmstadt LWE Challenge. The experiment result shows that our ProPnjBKZ is 7.6∼12.9 times more efficient than the heuristic uSVP solving algorithm in G6K. Besides, we break the TU Darmstadt LWE Challenges with $(n, \alpha) \in \{(40, 0.035), (40, 0.040), (50, 0.025), (55, 0.020), (90, 0.005)\}$.

Finally, we give a newly refined security estimator of LWE. The evaluation results indicate that the concrete hardness of the lattice-based NIST

⋆ Wenwen Xia and Leizhang Wang are the co-first authors of this work.

candidate schemes from LWE primal attack will decrease by 1.9∼4.2 bits when using our optimized blocksize and jump selection strategy and two-step solving mode. In addition, when using the list-decoding technology proposed by MATZOV in 2022, it further decreased by 8∼10.7 bits.

Keywords: cryptanalysis · lattice reduction · uSVP · progressive BKZ · PnjBKZ Simulator · optimized blocksize and jump strategy selection.

1 Introduction

To date, many post-quantum cryptosystems are lattice-based, e.g. Dilithium [1], Kyber [2] which have been accepted as NIST standards. Lattice-based structures appear to be immune from both classical and quantum attacks. As a result, many lattice-based constructions are considered secure, assuming that certain well-studied computational lattice problems cannot be solved in polynomial time. A large fraction of lattice-based cryptographic mechanisms are built upon the LWE problem [3] and its variants [2–5]. One of the best-known cryptanalytic techniques against these problems is primal attack [6], which is widely used in cryptanalysis of lattice-based cryptosystems. The primal attack solves the LWE problem by reducing it to the *unique Shortest Vector Problem* (uSVP$_\gamma$). Given a lattice $\mathcal{L}$ such that $\lambda_2(\mathcal{L}) > \gamma\lambda_1(\mathcal{L})$, the goal of uSVP$_\gamma$ is to find the unique lattice vector $\mathbf{v} \in \mathcal{L}$ with length $\lambda_1(\mathcal{L})$, which can be viewed as a variant of the approximate *Shortest Vector Problem* (SVP$_\gamma$) and solved through lattice reduction. Inputting an initial lattice basis, the lattice reduction algorithm outputs nearly orthogonal and short lattice vectors.

In recent years, substantial improvements have been made in lattice reduction algorithms. In 1982, the first polynomial-time lattice reduction algorithm named LLL [7] was proposed to solve SVP$_\gamma$ with an exponential approximate factor γ. To solve the problem with a smaller approximate factor, Schnorr and Euchner [8] presented *Block Korkin-Zolotarev reduction* (BKZ), which is considered as a combination of the LLL algorithm and the enumeration algorithm to balance the algorithm's time consumption and the success probability using a parameter β called blocksize. In the literature, many cryptanalysts improved the BKZ algorithm, e.g. the extreme pruning technique [9] to speed up enumeration, BKZ 2.0 [10] based on [9], approximate enumeration oracle [11], and parameters optimization in BKZ such as *Improved Progressive BKZ* (ProBKZ) [12].

However, solving uSVP$_\gamma$ is not exactly the same as solving SVP$_\gamma$. In [13], the authors gave a successful condition for the BKZ algorithm in solving uSVP$_\gamma$ which is verified by the experiments results of [14]. By the claim of [13, 14], the unique shortest vector is recovered by first finding the shortest vector in a projected sublattice and lifting it into the full lattice, which means that the reduced lattice basis is not the only requirement in solving uSVP$_\gamma$. A natural question arises as to whether there exists a better algorithm for solving uSVP$_\gamma$ than using only lattice-reduction algorithms such as BKZ.

General Sieve Kernel (G6K). In 2019, Albrecht *et al.* [15] designed the *General Sieve Kernel* (G6K), implemented the progressive sieving algorithm

named `Pump` which can selectively call the Gauss sieve [16, 17], NV sieve [18], k-list sieve [19, 20] or BGJ1 sieve [21]. `Pump` is a generic design based on sieving algorithms using the progressive sieve introduced in [22] (Similar ideas are also independently proposed in [23]) with dimension-for-free (d4f) technique [23], which makes the sieving process more efficient and allows a higher solving rate. Ducas *et al.* [24] improved the efficiency of G6K using GPU and implemented the fastest sieving algorithm BDGL16 [25] in both G6K and G6K-GPU-Tensor.

In addition to sieving algorithms, a new version of BKZ named *Pump-and-jump BKZ* (PnjBKZ) is also implemented in G6K. Unlike classical BKZ using an enumeration algorithm as its SVP oracle, PnjBKZ adopts `Pump` as its SVP oracle with a selective parameter `jump`. The `jump` value controls the jump stage of blocks in BKZ with the sieve oracle, which can jump by more than one dimension.

Solving uSVP$_\gamma$ in G6K. G6K provides an algorithm for solving LWE using primal attack which translates LWE to an uSVP$_\gamma$, and this LWE solving algorithm is a combination of PnjBKZ and Pump. The main idea is that each time after running a PnjBKZ-β tour with a large enough blocksize β, a `Pump` on a projected sublattice with similar running time (which means that the dimension of projected sublattice is slightly larger than β) is inserted into the PnjBKZ procedure. It then lifts the shortest vector in the projected sublattice onto the full lattice and checks whether it is the unique shortest vector.

It was shown by experiments that the new strategy of inserting `Pump` into PnjBKZ can improve the success rate in solving uSVP$_\gamma$ and LWE significantly. For the default parameter selection in G6K, it solves TU Darmstadt LWE Challenges 400 times faster than the previous records for comparable instances.

However, we show that the default mode in G6K still has a weakness, as the inserted `Pump` is not guaranteed to find the unique shortest vector. If the `Pump` failed, the G6K returns to the PnjBKZ procedure to further reduce the lattice basis. But since a large dimensional `Pump` is very inefficient for lattice basis reduction, the running time of a failed `Pump` is mostly wasted. The problem can be solved by giving an estimation of the success rate of the inserted Pump on a well-reduced lattice basis. To solve this problem we rely on a BKZ simulator with high accuracy to simulate the quality of lattice basis after PnjBKZ procedure.

BKZ Simulator. A BKZ simulator is used to predict the practical behavior of a BKZ algorithm (when $\beta \geq 45$), which is important in optimizing the parameter selection in BKZ. Based on the Gaussian heuristic, Chen and Nguyen refined the sandpile model from [26] and provided a BKZ simulator in BKZ 2.0 [10]. Using the properties that the last β vectors in BKZ-β reduced basis satisfy HKZ reduction and Gaussian Heuristic, [12] proposed a simulator for predicting BKZ-β fully reduced basis. Since the BKZ 2.0 simulator could not accurately predict the head concavity phenomenon after multiple tours of BKZ-β, Bai *et al.* [27] considered the norm of the shortest vector as a random variable rather than a fixed value, and brought randomness into the BKZ 2.0 simulator. The new simulator [27] can effectively predict and explain the phenomenon of head concavity of lattice basis reduced by BKZ. However, as above simulators did not consider the case of `jump`>1, so they cannot be directly used for PnjBKZ simulation.

Based on the BKZ simulator, in 2016, Aono *et al.* presented improved progressive BKZ (ProBKZ) [12]. For solving SVP_γ, ProBKZ calls a series of BKZ tours with different blocksizes to find the approximate shortest vector. One of the main contributions of their work is a blocksize strategy selection algorithm to generate the different blocksizes to be used in the BKZ reduction, which uses the shortest path algorithm to solve an optimized blocksize strategy by setting multiple different middle reduction qualities as the inner nodes. By using the blocksize strategy selection algorithm along with pre-processing and post-processing procedures, their ProBKZ runs faster than BKZ 2.0 which has a fixed blocksize. However, ProBKZ only considers enumeration as its subroutine, without using the more efficient lattice sieving algorithm. Besides, in this paper, we shall show that their method is not supposed to generate an optimal blocksize strategy, and still has room for improvement.

LWE Estimation. In addition to solving specific lattice hard problems, security estimation of lattice-based cryptography schemes is also one of the important applications of the BKZ algorithm. In 2022, three over four PQC candidates selected by NIST are lattice-based schemes [28]. In their documentations, they use leaky-LWE-Estimator to give refined security estimations that consider recent progress in lattice-based cryptanalysis compared with the conservative core-SVP model. Specifically, the leaky-LWE-Estimator first uses the estimator in [29] to calculate the expected value of BKZ blocksize of solving LWE. Then, it considers the influence of dimension-for-free technology [23] to calculate the total number of logic circuit gates needed to solve LWE by calling the gate-count algorithm proposed in [30]. However, in this paper, we show that their model will lead to an over-optimistic estimation. To ensure the security of these lattice-based PQC candidate schemes, it is necessary to evaluate the impact of our optimized blocksize and jump selection strategy and the two-step solving mode on the security of these lattice-based schemes.

Contribution. In this work, we formally define a two-step mode of solving uSVP_γ, which is divided into a reduction step and a searching step. The same idea has been proposed in [12] for solving SVP_γ with small approximate factor $\gamma = 1.05$, but it was not known previously to be applicable for uSVP_γ or LWE.

In particular, we prove that the two-step mode of solving uSVP_γ is more efficient than that of using BKZ reduction only. Besides, to increase the efficiency of the reduction step, we propose a blocksize and jump strategy selection algorithm to replace the trivial progressive reduction strategy in [31]. Our two-step mode of solving uSVP_γ and the optimal blocksize and jump strategy selection algorithm mainly relies on the accurate time cost models and simulating algorithms for PnjBKZ and `Pump`. More specifically:

- We propose a new two-step mode for solving uSVP_γ by simulating algorithms. We modify the default mode in G6K to a two-step mode, which firstly calls a series of PnjBKZs following a blocksize selection strategy to reduce the basis and then uses a `Pump` algorithm to search the unique shortest vector. Compared to the G6K's default strategy, we can efficiently save time cost of improving the quality of lattice basis through the new blocksize and jump strategies.

Besides, we avoid wasting time due to failed Pumps in G6K's default strategy by a high solving probability `Pump` which executes only once. Eventually, we significantly improve the efficiency of solving uSVP$_\gamma$ and it becomes faster than that of G6K's default strategy through the improvement during these two steps. We are not only the first who give a two-step lattice solving algorithm for BKZ with lattice sieving, but also the first who give a theoretical analysis on the parameter selection of a two-step mode based on a BKZ simulator.

- We construct a new simulator to simulate the PnjBKZ reduction, especially in the case of jump > 1, and give a new `Pump` estimation algorithm for determining the dimension in `Pump` when solving LWE problem. Furthermore, we design a new simulator to simulate the quality of lattice basis after Pump. Both our PnjBKZ simulator and `Pump` simulator are verified experimentally.

- Based on the simulating algorithms and two-step mode, to improve the efficiency of the reduction step, we give two new reduction strategy generation algorithms, which generate the blocksize and jump strategy and the suitable sieving dimension for `Pump` to minimize the expected cost of solving uSVP$_\gamma$. The code for strategy selection algorithms for solving LWE is available in the github[1]. We borrow the same shortest path algorithm as in ProBKZ to design our first algorithm called *blocksize strategy selection algorithm based on ProBKZ* (BSSA) by replacing the BKZ and enumeration algorithm with PnjBKZ and `Pump` respectively. However, we find that the strategy generated by BSSA has room for improvement. To obtain a reduction strategy to minimize the simulated cost for ProPnjBKZ, we design a new strategy selection algorithm named *blocksize strategy enumeration* (EnumBS). EnumBS can obtain a better blocksize and jump selection strategy at a higher theoretical complexity than BSSA when generating, but the time is still acceptable for low-dimensional lattices.

- We test the performance of our two-step mode of solving uSVP$_\gamma$ based on our blocksize and jump strategy selection algorithm. Using the blocksize strategy chosen from EnumBS, the algorithm significantly increases the efficiency at most 12.9 times (at least 7.6 times) in solving the TU Darmstadt LWE Challenges compared with that of the default LWE solver in G6K shown in Table 4 in Sec. 6.2. We have uploaded the executable code of our ProPnjBKZ[2] in the github. Besides, we cracked the TU Darmstadt LWE Challenges[3] $(n, \alpha) \in \{(40, 0.035), (90, 0.005), (50, 0.025), (55, 0.020), (40, 0.040)\}$ for the first time.

- Based on the leaky-LWE-Estimator, our optimized blocksize and jump selection strategy, and the two-step solving mode, we propose a newly refined security estimator for estimating the concrete hardness of LWE[1]. Meanwhile, evaluation results in Table 7 in Sec. 6.4 indicate that compared with the estimation in [28], by considering our optimized blocksize and jump selection strategy, and the two-step solving mode to attack, the concrete hardness of these lattice-based NIST PQC candidates under primal attack for LWE will decrease by 1.9∼4.2 bits (8.0∼10.7 bits if using list-decoding skill [32] shown in Table 11 in Appendix G).

[1] https://github.com/Summwer/lwe-estimator-with-pnjbkz.git
[2] https://github.com/Summwer/pro-pnj-bkz
[3] https://www.latticechallenge.org/lwe_challenge/challenge.php

Organization. The paper is organized as follows. Sec. 2 presents the basic notations and preliminaries. Sec. 3 gives a sketch of ProPnjBKZ. Sec. 4 design a PnjBKZ simulator and a `Pump` sieving dimension estimator for simulation. We give an detailed description on our two blocksize strategy selection algorithms in Sec. 5. Besides, we compare their costs on solving LWE instances in Sec. 6. Meanwhile, Sec. 6 also gives a new refined security estimation of NIST PQC schemes also shown. Finally, we give conclusion and further work in Sec. 7.

2 Preliminaries

2.1 Notations and Basic Definitions

We write a matrix $\mathbf{B}$ as $\mathbf{B} = (\mathbf{b}_0, \cdots, \mathbf{b}_{d-1})$ where $\mathbf{b}_i$ is the $(i+1)$-th column vector of $\mathbf{B}$. The Euclidean norm of a vector $\mathbf{v}$ is denoted by $\|\mathbf{v}\|$. If $\mathbf{B} \in \mathbb{R}^{d\times d}$ has full rank d, the lattice $\mathcal{L}$ generated by the basis $\mathbf{B}$ is denoted by $\mathcal{L}(\mathbf{B}) = \{\mathbf{Bx}|\mathbf{x} \in \mathbb{Z}^d\}$. We denote $\mathbf{B}^* = (\mathbf{b}_0^*, \ldots, \mathbf{b}_{d-1}^*)$ as the Gram-Schmidt orthogonalization of $\mathbf{B}$, in which $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=0}^{i-1} \mu_{i,j}\mathbf{b}_j^*$, $\mu_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle / \|\mathbf{b}_j^*\|^2$, for $i \in \{0, \cdots, d-1\}$. Let the orthogonal projection to the span of $(\mathbf{b}_0, \cdots, \mathbf{b}_{i-1})$ be π_i, i.e. $\forall \mathbf{v}$, $\pi_i(\mathbf{v}) = \mathbf{v} - \sum_{j=0}^{i-1} \omega_j \mathbf{b}_j^*$, where $\omega_j = \langle \mathbf{v}, \mathbf{b}_j^* \rangle / \|\mathbf{b}_j^*\|^2$. For $i, j \in \mathbb{Z}_d$ and $0 \le i < j \le d-1$, given an arbitrary d-dimensional vector $\mathbf{v} = (v_0, \cdots, v_{d-1})$, define $\mathbf{v}_{[i:j]}$ as $(v_i, \cdots, v_{j-1})$ with a size $j-i$. For a lattice basis $\mathbf{B}$, let $\mathbf{B}_{[i:j]} \leftarrow (\mathbf{b}_i, \cdots, \mathbf{b}_{j-1})$. Moreover, we denote $\mathbf{B}_{\pi[i:j]}$ by the local projected block $(\pi_i(\mathbf{b}_i), \cdots, \pi_i(\mathbf{b}_{j-1}))$, and call $\mathcal{L}_{\pi[i:j]}$ the lattice generated by $\mathbf{B}_{\pi[i:j]}$. We use $\mathbf{B}_{\pi[i]}$ and $\mathcal{L}_{\pi[i]}$ as shorthands for $\mathbf{B}_{\pi[i:d]}$ and $\mathcal{L}_{\pi[i:d]}$. The volume of a lattice $\mathcal{L}(\mathbf{B})$ is $\mathrm{Vol}(\mathcal{L}(\mathbf{B})) = \prod_{i=0}^{d-1} \|\mathbf{b}_i^*\|$, an invariant of the lattice. The first minimum of a lattice $\mathcal{L}(\mathbf{B})$ is the length of the shortest non-zero vector, denoted by $\lambda_1(\mathcal{L}(\mathbf{B}))$. We use the abbreviations $\mathrm{Vol}(\mathbf{B}) = \mathrm{Vol}(\mathcal{L}(\mathbf{B}))$ and $\lambda_1(\mathbf{B}) = \lambda_1(\mathcal{L}(\mathbf{B}))$.

Notations for algorithms description. Let BKZ-β/PnjBKZ-(β, J) be an abbreviation of a one-tour BKZ/PnjBKZ with blocksize β and jump value J. Assume $\mathbf{B} = (\mathbf{b}_0, \cdots, \mathbf{b}_{d-1})$, its Gram-Schmidt basis is $\mathbf{B}^* = (\mathbf{b}_0^*, \cdots, \mathbf{b}_{d-1}^*)$. Denote l_i by the logarithm of Gram-Schmidt norm, i.e. $l_i = \ln(\|\mathbf{b}_i^*\|)$, for $i \in \{0, \cdots, d-1\}$. Let $\mathsf{rr}(\mathbf{B}) = (l_0, \cdots, l_{d-1})$, abbreviate to rr, $\mathsf{rr}_{[i:j]} = (l_i, \cdots, l_{j-1})$.

Denote BKZSim by the BKZ simulator proposed in [10]. The simulation for PnjBKZ is denoted as PnjBKZSim$(\mathsf{rr}(\mathbf{B}(\mathsf{rr}), \beta, J, t)$, which simulates a PnjBKZ-(β, J) with t tours on the lengths rr and return the new lengths. Moreover, if we have a blocksize and jump strategy S that stores a series of (β_i, J_i), then PnjBKZSim$(\mathsf{rr}, \mathsf{S})$ means iteratively calling a tour of PnjBKZ-(β_i, J_i) simulator on rr, where $(\beta_i, J_i) \in \mathsf{S}$. Assume the input basis is $\mathbf{B}$, and the basis $\mathbf{B}$ reaches a basis quality after calling sufficient tours of BKZ-β. To simplify the above step, we use β to imply the quality of a BKZ-β reduced basis. Let $\sharp$tours(BKZ-β)/$\sharp$tours(PnjBKZ-(β, J)) be the minimum tours for BKZ-β/PnjBKZ-(β, J) to reach a BKZ-β/PnjBKZ-(β, J) reduced basis, abbreviated as $\sharp$tours. Denote t as the number of tours for implementing BKZ/PnjBKZ with a fixed blocksize β.

Let $T_{\mathrm{BKZ}}(\beta)$/$T_{\mathrm{PnjBKZ}}(\beta, J)$ be the time cost of one-tour BKZ/PnjBKZ with blocksize β and jump value J. Let $T_{\mathrm{PnjBKZs}}(\mathsf{S})$ be the total time cost for a series

of PnjBKZ with a specific reduction strategy $\mathsf{S}=\{(\beta_0, J_0), \cdots, (\beta_{n-1}, J_{n-1})\}$, abbreviate it as T_{PnjBKZs}. Denote $T_{\text{Pump}}(d_{\text{svp}})$ as the time cost of `Pump` with d_{svp} sieving dimension, abbreviate it as T_{Pump}. Let PSC be the expected `Pump` cost to find the target vector, which will be explained in the Sec. 4.2.

Definition 1. *(The Gaussian Distribution [33]) Let $\sigma, u \in \mathbb{R}$ be the standard deviation and the mean value respectively, a continuous Gaussian Distribution denoted as $N(u, \sigma^2)$. Its probabilistic density function $\rho_{N(u,\sigma^2)} = e^{-\frac{(x-u)^2}{2\sigma^2}} / \sigma\sqrt{2\pi}$.*

Definition 2. *(Chi-Squared Distribution [33]) Given n random variables $X_i \sim N(0,1)$, the random variables $X_0^2 + \cdots + X_{n-1}^2$ follows a chi-squared distribution χ_n^2 over $\mathbb{R}^*$ of mean n and variance $2n$ with probabilistic density function $\rho_{\chi_n^2}(x) = x^{\frac{n}{2}-1} e^{-\frac{x}{2}} / 2^{\frac{n}{2}} \Gamma(n/2)$. Given n random variables $Y_i \sim N(0, \sigma^2)$, the random variables $Y_0^2 + \cdots + Y_{n-1}^2$ follows a scaled chi-squared distribution $\sigma^2 \cdot \chi_n^2$ over $\mathbb{R}^*$ of mean $n\sigma^2$ and variance $2n\sigma^2$.*

Heuristic 1 *(Gaussian Heuristic [23]) The expected first minimum of a lattice $\mathcal{L}$ (denoted as $\lambda_1(\mathcal{L}(\mathbf{B}))$) according to the Gaussian Heuristic denoted by $\mathrm{GH}(\mathcal{L})$ is given by $\lambda_1(\mathcal{L}(\mathbf{B})) \approx \mathrm{GH}(\mathcal{L}) = \left(\Gamma(\frac{d}{2}+1) \cdot \mathrm{Vol}(\mathcal{L})\right)^{\frac{1}{d}} / \sqrt{\pi} \approx \sqrt{d/(2\pi e)} \cdot \mathrm{Vol}(\mathcal{L})^{\frac{1}{d}}$ Where $\mathrm{V}_d(1)$ is the volume of the d-dimensional unit sphere. We also write $\mathrm{GH}(\mathbf{B}) = \mathrm{GH}(\mathcal{L}(\mathbf{B}))$ and $\mathrm{GH}(\mathsf{rr}_{[i:j]}) = \mathrm{GH}(\mathbf{B}_{\pi[i:j]})$.*

Definition 3. *(HKZ reduction and BKZ reduction [23]) The basis $\mathbf{B}$ of a lattice $\mathcal{L}$ is HKZ reduced if $\mathbf{b}_i^* = \lambda_1(\mathcal{L}(\mathbf{B}_{\pi[i:d]}))$, for all $i < d$. $\mathcal{L}$ is BKZ-β reduced if $\mathbf{b}_i^* = \lambda_1(\mathcal{L}(\mathbf{B}_{\pi[i:\min\{i+\beta,d\}]}))$, for all $i < d$.*

Definition 4. *(Root Hermite Factor [34]) For a basis $\mathbf{B}$ of d-dimensional lattice, the root Hermite factor is defined as $\delta = \left(\|\mathbf{b}_0\| / \mathrm{Vol}(\mathbf{B})^{1/d}\right)^{1/d}$, for estimating the equality of the output vector of BKZ. For larger blocksize, it follows the asymptotic formula $\delta(\beta)^{2(\beta-1)} = \frac{\beta}{2\pi e}(\beta\pi)^{1/\beta}$.*

Heuristic 2 *(Geometric Series Assumption [15]) Let $\mathbf{B}$ be a lattice basis after lattice reduction, then Geometric Series Assumption states that $\|\mathbf{b}_i^*\| \approx \alpha \cdot \|\mathbf{b}_{i-1}^*\|$, $0 < \alpha < 1$. Combine the GSA with root-Hermite factor (Definition 4) and $Vol(\mathcal{L}(\mathbf{B})) = \prod_{i=0}^{d-1} \|\mathbf{b}_i^*\|$, it infers that $\alpha = \delta^{-\frac{2d}{d-1}} \approx \delta^{-2}$.*

2.2 Lattice Hard Problems

Definition 5. *(unique Shortest Vector Problem($uSVP_\gamma$) [35]) Given an arbitrary basis $\mathbf{B}$ on lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, $\mathcal{L}$ satisfies the condition $\gamma\lambda_1(\mathbf{B}) < \lambda_2(\mathbf{B})$ ($\gamma > 1$, $\lambda_2(\mathbf{B})$ is norm of the second shortest vector which is linearly independent to the shortest vector), find the shortest non-zero vector $\mathbf{v}$ s.t. $\|\mathbf{v}\| = \lambda_1(\mathbf{B})$.*

Definition 6. *(LWE_{m,n,q,D_σ} Distribution [36–38]) Given some samples $m \in \mathbb{Z}$, a secret vector length $n \in \mathbb{Z}$, a modulo $q \in \mathbb{Z}$, a probability distribution D_σ. Uniformly sample a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and sample a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ from a specific distribution, randomly sample a relatively small noise vector $\mathbf{e} \in \mathbb{Z}_q^m$ from Gaussian distribution D_σ whose standard deviation is σ. The LWE distribution Ψ is constructed by the pair $(\mathbf{A}, \mathbf{b} = \mathbf{As} + \mathbf{e}) \in (\mathbb{Z}_q^{m \times n}, \mathbb{Z}_q^m)$ sampled as above.*

Definition 7. *(Search LWE_{m,n,q,D_σ} problem [36–38]) Given a pair* $(\mathbf{A}, \mathbf{b})$ *sampled from LWE distribution* Ψ *compute the pair* $(\mathbf{s}, \mathbf{e})$.

2.3 G6K and G6K-GPU-Tensor

G6K [15] is an abstract machine for running sieve and reduction algorithms, which is built on generalizing and extending the previous sieve algorithms. G6K-GPU-Tensor as a state-of-art SVP solver improves the efficiency of G6K by GPU implementations, and holds many records in TU Darmstadt SVP Challenges.

2.4 Sieving Algorithms and technologies in G6K

Sieving Algorithms The first practical NV sieving algorithm uses a database of $N_0 \approx 2^{0.2075d+o(d)}$ vectors and runs in time $N_0^2 \approx 2^{0.415d+o(d)}$ by repeatedly checking all pairs $\mathbf{v} \pm \mathbf{w}$ [18]. To find the shortest vector, N_0 is the minimal number of vectors to ensure saturating the ball of radius $\mathrm{GH}(L)\sqrt{4/3}$ by short vector. In a line of works [21, 25, 39, 40] the time complexity was gradually decreased to $2^{0.292d+o(d)}$ by nearest neighbour searching techniques.

Progressive Sieve Progressive sieve [23] can save the cost of classical sieve. It realized by a right-to-left operation which first calls a sieve on a small dimension projected lattice, then uses Babai's nearest plane algorithm [41] to recover the vector to a higher dimension projected lattice. Repeat such step until recover the short vectors onto a full dimensional lattice.

Dimension for Free (d4f) Technique D4f technology [23] can bring sub-exponential time speedup and memory decreasing for sieve algorithms. [23] has given two theoretical d4f estimations for solving β-dimension SVP as $\mathrm{d4f}(\beta) = \beta\ln(4/3)/\ln(\beta/2\pi)$ and $\mathrm{d4f}(\beta) = \beta\ln(4/3)/\ln(\beta/2\pi e)$, while in the implementation of G6K [15], it gives a more relaxed bound $\mathrm{d4f}(\beta)$: when $\beta < 40$, $\mathrm{d4f}(\beta) = 0$; when $40 \leq \beta \leq 75$, $\mathrm{d4f}(\beta) = \lfloor \beta - 40/2 \rfloor$; when $\beta > 75$, $\mathrm{d4f}(\beta) = \lfloor 11.5 + 0.075\beta \rfloor$.

Pump in G6K Albrecht *et al.* proposed `Pump` algorithm in [15], which is improved based on *Progressive Sieve* [22] with d4f technique [23] and the insertion tricks in [15]. There are four input parameters for `Pump` algorithm: lattice basis $\mathbf{B}$, left insertion bound κ, insertion upper bound d_{svp} and d4f value $\mathrm{d4f}(d_{\mathrm{svp}})$. Here $\kappa + d_{\mathrm{svp}} = d$ and the upper bound of sieve dimension is $d_{\mathrm{svp}} - \mathrm{d4f}(d_{\mathrm{svp}})$.

Slope in G6K To measure the quality of lattice basis [15] use the averaged quality measurement. It is the least squares fit coefficient of the slope of $\log\|\mathbf{b}_i^*\|$, which means that the slope closer to 0, the better basis quality.

2.5 PnjBKZ in G6K

PnjBKZ is a BKZ-type reduction algorithm that uses `Pump` as its SVP oracle. Unlike classical BKZ, PnjBKZ performs the SVP oracle with an adjustable jump no less than 1. Specifically, runing a PnjBKZ with blocksize β and jump=J, after executing the SVP oracle on a certain block $\mathbf{B}_{[i:i+\beta]}$, the next SVP oracle will be executed on the $\mathbf{B}_{[i+J:i+\beta+J]}$ block with a jump count J rather than $\mathbf{B}_{[i+1:i+\beta+1]}$.

3 ProPnjBKZ in Two-step mode for solving uSVP$_\gamma$

In this section, we firstly give a comparison among BKZ-only mode, the default mode in G6K and two-step mode (Fig. 1) to explain the benefit of two-step mode in Sec. 3.1. Then, we give a sketch of ProPnjBKZ in Sec. 3.2, where we use the two-step mode with a strategy selection method for solving uSVP$_\gamma$.

3.1 Comparison of uSVP$_\gamma$ solving Mode

In this part, we introduce BKZ-only Mode and Default Mode in G6K for solving the uSVP$_\gamma$ in the literature and compare them with our two-step mode. We also give an experiment to prove the comparison result.

BKZ-only Mode. BKZ-only mode [13, 34] (Fig. 1(a)) implements multiple tours of BKZ-β/PnjBKZ-$(\beta, J = 1)$ whose blocksize is fixed until solving the uSVP$_\gamma$ problem. The BKZ-only mode is the mainstream method in the security estimation of LWE-based cryptosystems, such as in [29].

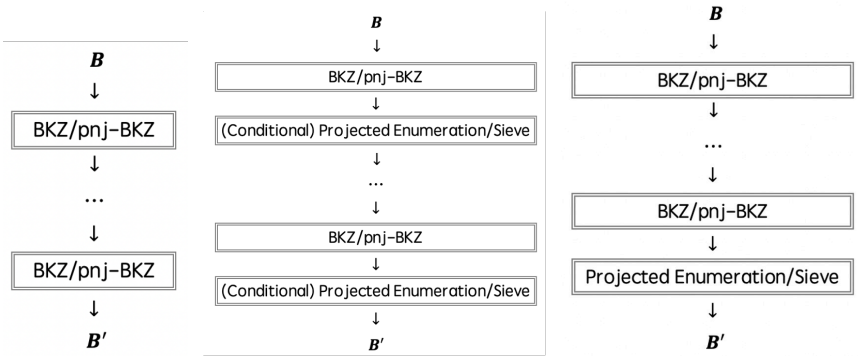
Default uSVP$_\gamma$ solving Mode in G6K G6K [15,24] solve the LWE problem by primal attack, i.e. reduce LWE to the uSVP$_\gamma$ and solve it by calling progressive PnjBKZ and a conditional `Pump` (The algorithm calls `Pump` only if the estimated time cost of `Pump` is shorter than an upper bound computed) repeatedly. We can extract a uSVP$_\gamma$ solving mode from their implement `lwe_challenge.py` and name it as the *Default uSVP$_\gamma$ solving Mode in G6K* (Fig. 1(b)).

In the default uSVP$_\gamma$ solving mode of G6K, it will reduce the basis by a specific blocksize strategy S_0[4]. After each lattice reduction by PnjBKZ-$(\beta, J = 1)$, $\beta \in \mathsf{S}_0$, the default uSVP$_\gamma$ solving mode will record the time cost of the reduction process and determine whether a `Pump` will finish in the same cost. If it does, it will call a `Pump`; If not, it will skip to the next PnjBKZ-$(\beta, J = 1)$.

The benefit of the default uSVP$_\gamma$ solving mode in G6K is that if we do not have an accurate simulator for BKZ/PnjBKZ and we are not sure of the solvability by a final `Pump` calling, then a Default uSVP$_\gamma$ solving Mode in G6K will make sure in outputting the required result in a reasonable time. However, without a simulator, it will sometimes enter a `Pump` with solving failure and waste processing time heavily since a `Pump` call is costly. Here a failed `Pump` means that

[4] $\mathsf{S}_0 = \text{list}(\text{range}(10, 50)) + [\text{b} - 20, \text{b} - 17] + \text{list}(\text{range}(\text{b} - 14, \text{b} + 25, 2))$.

the dimension setting of the sieving in the Pump is over-optimistic, which makes the Pump fail to find the target vector. Besides, it might enter a Pump late and waste the processing time of extra cost for several PnjBKZs with large blocksizes.



(a) BKZ-only (b) Default Mode in G6K (c) Two-step

Fig. 1. Different mode to solve the uSVP$_\gamma$ Problem: (a) BKZ-only Mode; (b) Default uSVP$_\gamma$ Solving Mode ing G6K; (c) Two-step Mode.

Two-step Mode The two-step mode (Fig. 1(c)) was first informally stated in [12], in the term called "lattice-based attack" for solving exact SVP ($\gamma = 1.05$), which calls a series of BKZ first for lattice reduction and calls an enumeration algorithm to find the shortest vector at last (called point search). However, their algorithm is not known to be applicable for uSVP$_\gamma$.

In this paper, we show that a two-step mode adapted to PnjBKZ and Pump is more efficient in solving uSVP$_\gamma$. Our algorithm calls a series of PnjBKZ for reduction first and at a good timing uses a Pump algorithm to search the unique shortest vector. By our PnjBKZ simulator Alg. 2 and Pump sieving dimension estimation Alg. 3 in Sec. 4.1, we ensure that the last Pump outputs target vector.

Experiments of Comparison among BKZ-only Mode, Default Mode in G6K, and Two-step Mode In this part, we give an experimental result to illustrate that for solving uSVP$_\gamma$, the two-step mode is better than both BKZ-only mode and G6K default mode. For comparison, assume that the blocksize strategy used in different modes are the same and simply let jump $= 1$.

The two-step mode and the BKZ-only mode both run the same BKZ tours at the beginning. However, in their final stages to find the unique shortest vector, the two-step mode calls a Pump, while the BKZ-only mode calls more BKZ tours. We show that Pump is more efficient in recovering short lattice vector than BKZ.

We call a Pump-$(\kappa, d_{\text{svp}}, f)$ and a PnjBKZ-(β, J) ($\beta < d_{\text{svp}}$) separately on the same lattice basis, denote cost of Pump as T_{Pump}. Table 1 shows that the reduced

shortest vector $\mathbf{b}_0$ after a Pump is shorter than that after a PnjBKZ-($\beta = 55, J = 1$) while $T_{\text{Pump}} \leq T_{\text{PnjBKZ}}$. Thus, calling a Pump is more likely to find a shorter vector compared to the PnjBKZ in no more time cost than PnjBKZ. Therefore, it states that two-step mode is better than the BKZ-only mode.

On the other hand, both the two-step mode and the G6K default mode end with a Pump which outputs the unique shortest vector. Their differences are at the earlier stage, where the two-step mode always call BKZ, while the G6K default mode may call an early Pump without a solution.

Table 1. Simulated norm of $\mathbf{b}_0$ after a PnjBKZ and a Pump under the same time cost.

$(n,\alpha)^\dagger$	Cost‡	$\ln(\|\mathbf{b}_0\|^2)$	
		PnjBKZ	Pump($\mathcal{L}_{[\kappa:d]}$)
(55,0.005)	23.9	15.82	10.55
(40,0.015)	6.6	14.76	11.27
(45,0.010)	18.0	15.04	11.12
(40,0.020)	17.1	14.76	12.05

† Basis from LWE instance (n, α) in TU Darmstadt LWE Challenge.

Table 2. Basis quality estimation after a PnjBKZ and a Pump under the same time cost.

(n,α)	Cost‡	$\log_2(\text{PSC})(\log_2 \text{h})$	
		PnjBKZ	Pump($\mathcal{L}_{[\kappa:d]}$)
(55,0.010)	6.0	7.94	8.44
(60,0.005)	10.3	11.69	12.01
(70,0.005)	12.6	16.01	17.34
(75,0.005)	14.9	21.68	22.76

‡ The cost limit in minutes while calling the algorithm.

In the comparison between the two-step mode and G6K default mode, we show that an early Pump is less helpful in solving uSVP$_\gamma$ than an early PnjBKZ. Let the time cost of Pump for finding the target norm on the specific lattice basis, i.e. PSC, be a standard of measuring the quality of lattice basis reduced by different algorithms. Lattice basis with low PSC can be regarded as better quality. Our experiments by separately call a PnjBKZ/Pump on the same lattice basis. Table 2 shows that basis quality after a PnjBKZ-($\beta = 60, J = 5$) reduction is better than that after a Pump reduction while $T_{\text{PnjBKZ}} \leq T_{\text{Pump}}$. The latter basis quality is estimated by the PSC estimation Alg. 3. So, in the G6K-default mode, if it enters a Pump with no solution, the quality of the returned lattice basis will be worse than that after a PnjBKZ reduction under a same time limit. In conclusion, the G6K-default mode is less efficient than the two-step mode.

3.2 Algorithm overview

Our *Improved Progressive PnjBKZ*(ProPnjBKZ) are designed in two-step mode and a strategy selection algorithm, which will introduce in Sec. 5. It aims to speedup the efficiency for solving the uSVP$_\gamma$ and can be described as the following: input a lattice basis $\mathbf{B}$, and the distribution function $F(\star, \mathcal{D})$ of the (projected) target norm (see Sec. 4.2). It first generates an optimized blocksize and jump strategy S using a strategy selection algorithm (EnumBS or BSSA, see Sec. 5) and then reduce $\mathbf{B}$ through a series of PnjBKZ-(β, J), in which each (β, J) is selected from S. To minimize the total cost, it will first use a series of PnjBKZ to reduce the lattice basis properly, then at the suitable timing, it will call a Pump algorithm to find the target vector. The parameter selection of Pump follows Alg. 3, which will lead to finding the unique shortest vector after Pump. The detailed process is as Alg. 1.

input : $\mathbf{B}$, $F(\star, \mathcal{D})$;
output: The approximate shortest vector $\mathbf{v}$;
Function ProPnjBKZ($\mathbf{B}$, $F(\star, \mathcal{D})$):
  $\mathbf{B}$ = LLL($\mathbf{B}$);
  Generate Strategy S using EnumBS or BSSA;
  for $(\beta, J, \sharp\text{tours}) \in$ S **do**
    for t **from** 1 **to** $\sharp$tours **do**
      $\mathbf{B} \leftarrow$ PnjBKZ($\mathbf{B}, \beta, J, \sharp$tours);
  $d_{\text{svp}}, \_ \leftarrow$ ProSieveDimEst(rr($\mathbf{B}$), $F(\star, \mathcal{D})$); $f \leftarrow$ d4f(d_{svp});
  $\mathbf{B} \leftarrow$ Pump($\mathbf{B}$,$d - d_{\text{svp}}$, d_{svp}, f);
  return $\mathbf{v} \leftarrow \mathbf{b}_0$;

Algorithm 1: Improved Progressive PnjBKZ

Our algorithm improves the heuristic uSVP$_\gamma$ solver in G6K from two aspects: firstly, we use a two-step mode with an optimal blocksize strategy; secondly, we choose the optimized jump($\geq$1) strategy instead of always keeping the jump=1.

Although the experiments in [31] suggest that compared with the reduction strategy of jump=1, the reduction strategy of jump>1 is not beneficial, we show that it is not the case. More precisely, [31] shows that the reduction strategy of jump=3 requires similar running time to obtain the same quality of lattice basis reduced by the strategy of jump=1, with a larger memory consumption. However in our experiments, while the jump strategy becomes larger, the walltime for reaching the same lattice basis quality decreases significantly. Our experiment result shows that compared with the experiments results in [31], using a more efficient parallel computing implementation in [24], the acceleration effect of the strategy with jump>1 on the improvement of lattice quality is indeed more obvious. More details can be found in Fig. 2, which shows that the PnjBKZ with the jump>1 has a smaller time cost (3$\sim$6 times faster) while achieving the same reduction quality as that of the PnjBKZ with jump=1. Therefore, to find the optimal PnjBKZ reduction parameters, it is essential to construct a PnjBKZ simulator to handle the case for jump>1 which we will discuss in Sec. 4.1.

4 Simulators in Two-Step Mode of Solving uSVP$_\gamma$

In this section, we construct the simulators we need to design the optimized Blocksize and Jump Strategy, to obtain the expected minimal cost of the two-step mode of solving uSVP$_\gamma$.

Specifically, in Sec. 4.1, we first give the construction of the PnjBKZ simulator and show the validation experiments of the performance of the PnjBKZ simulator. Then we give the Pump cost model and the sieving dimension estimation of the last Pump in two-step solving mode in Sec. 4.2.

Set Gram-Schmidt vectors reduced by one tour of BKZ-β and PnjBKZ-(β, J) respectively as $l_i^{'}$=ln($\|\mathbf{b}_i^{*'}\|$) and $l_i^{''}$=ln($\|\mathbf{b}_i^{*''}\|$), for $i \in \{0, \cdots, d-1\}$. Denote

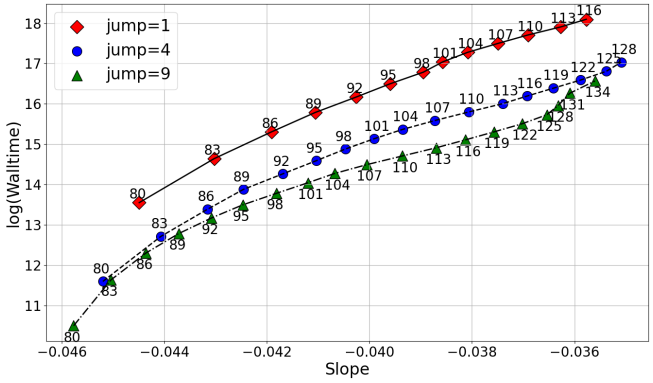


Fig. 2. Efficiency Speedup in Reduction by Jump strategy. Test on a 252-dimension lattice basis. The walltime and slope are averaged over 5 instances for each algorithm. Each instance ran on a machine with 2 GPUs, 32 threads. The points labeled by β.

BKZSim by the BKZ simulator proposed in [10]. Denote our PnjBKZ simulator as PnjBKZSim to simulate the change of lattice basis $\mathbf{B}$ after calling a PnjBKZ.

4.1 PnjBKZ Simulator

The first step in the two-step solving mode is using a series of well-chosen PnjBKZ-(β, J, t) to reduce the lattice basis. To find the optimized reduction strategy of PnjBKZ with $\mathsf{jump} > 1$ an accurate PnjBKZ simulator is necessary.

The PnjBKZ Simulator Construction Before we give the detailed construction of our PnjBKZ simulator, let's briefly review the main idea of the BKZ simulator proposed in [10]. The BKZ simulator proposed in [10] first will calculate $\mathrm{Sim}(l_0^{'}) = \ln\left(\mathrm{GH}(\mathcal{L}_{[0:\beta-1]})\right) \approx \frac{1}{2}\ln\left(\beta/2\pi e\right) + \ln\left(\mathrm{Vol}(\mathcal{L}_{[0:\beta-1]})\right)/\beta$ by Gaussian Heuristic (Heuristic 1). Then it will calculate $\mathrm{Sim}(l_1^{'}) = \ln\left(\mathrm{GH}(\mathcal{L}_{[1:\beta]})\right) \approx \frac{1}{2}\ln\left(\beta/2\pi e\right) + \left(\ln \mathrm{Vol}(\mathcal{L}_{[0:\beta]}) - \mathrm{Sim}(l_0^{'})\right)/\beta$ by Heuristic 1 and the information of $\mathrm{Sim}(l_0^{'})$ since the insert of new $\mathbf{b}_1$ will change the value of $\mathrm{Vol}(\mathcal{L}_{[1:\beta]}) = \prod_{i=1}^{\beta}\|\mathbf{b}_i^*\|$ to $\mathrm{Vol}(\mathcal{L}^{'}_{[1:\beta]}) = \left(\prod_{i=0}^{\beta}\|\mathbf{b}_i^*\|\right)/\|\mathbf{b}_0^{*'}\|$. Here $\mathrm{Sim}(l_0^{'})$ is a simulated approximate value of $l_0^{'}$ by Heuristic 1. Iteratively calculating all remaining unknown $\mathrm{Sim}(l_i^{'})$, such a simulator can predict the value of each $l_i^{'}$ in $\mathbf{B}^{*'}$.

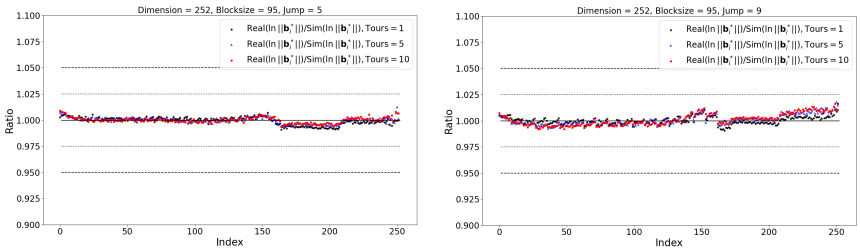
However, the BKZ 2.0 simulator [10] cannot be used directly to simulate the behavior of PnjBKZ when $\mathsf{jump}>1$. We observe that when $J>1$, each time after new $\mathbf{b}_i^*$ inserting at the first position of the block $\mathbf{B}_{\pi[i:i+\beta]}$, the $J-1$ norms of Gram-Schmidt vectors $\mathbf{b}_{i+1}^*, ..., \mathbf{b}_{i+J-1}^*$ will change and remain unknown. These unknown norms prevent the BKZ 2.0 simulator [10] from predicting the norm of the first Gram-Schmidt vector in the next block. Our idea is that when jump>1, we let each projected sub-lattice basis be reduced by `Pump` satisfying HKZ reduced so that we can predict these unknown norms between adjacent blocks.

Let l_i'' be the logarithm of each Gram-Schmidt norm after one tour of PnjBKZ-$(\beta,\ J)$. If the first J vectors in each block reduced by `pump` satisfies HKZ reduced[5], then we simulate each l_i after a PnjBKZ-$(\beta,\ J)$ reduction (denoted as $\mathrm{Sim}(l_i'')$) by Heuristic 1.

$$\mathrm{Sim}(l_i'') = \ln\left(\mathrm{GH}\left(\mathcal{L}''_{\pi[i:k]}\right)\right),\ k = \begin{cases} i-(i \bmod J)+\beta,\ i \in [0, d-\beta-1], \\ d,\ i \in [d-\beta, d-1]. \end{cases} \tag{1}$$

The key is how to calculate the volume of $\mathcal{L}''_{\pi[i:k]}$. Same as BKZ reduction during the process of PnjBKZ reduction the volume of $\mathcal{L}''_{\pi[0:k]}$ equals to that of $\mathcal{L}_{\pi[0:k]}$. Suppose we already know $\mathrm{Sim}(l_j'')$, $j \in \{0, ..., i-1\}$, we calculate $\ln\left(\mathrm{Vol}\left(\mathcal{L}''_{\pi[i:k]}\right)\right) = \ln\left(\mathrm{Vol}\left(\mathcal{L}_{[0:k]}\right)\right) - \ln\left(\mathrm{Vol}\left(\mathcal{L}''_{[0:i]}\right)\right) = \sum_{j=0}^{k} l_j - \sum_{j=0}^{i-1} \mathrm{Sim}(l_j'')$. Finally, based on Heuristic 1 we iteratively calculate $\mathrm{Sim}(l_i'')$ by Eq. (1).

In other words, we only need to input the initial Gram-Schmidt norms $l_i = \ln(\|\mathbf{b}_i^*\|)$, $i \in \{0, \cdots, d-1\}$ of the lattice basis. Without performing PnjBKZ reduction, we can simulate l_i'' by Eq. (1), which describes the change of lattice basis after each tour of PnjBKZ-$(\beta,\ J)$. Here l_i'' are these actual Gram-Schmidt vector norms of lattice base after reducing by one tour of PnjBKZ-$(\beta,\ J)$.



(a) Jump=5 (b) Jump=9

Fig. 3. ratio $l_i''/\mathrm{Sim}(l_i'')$, β=95. Run 10 tours of PnjBKZ-(95,5) and PnjBKZ-(95,9) reduction on a 252-dimension lattice basis, and record the output of Gram-Schmidt vector lengths each tour. We test 20 times for each reduction parameters.

To show that it is reasonable for us to use the properties of the HKZ reduction basis to simulate the actual reduction effect of PnjBKZ-$(\beta,\ J)$, we first illustrate that if J is below a specific upper bound (we heuristicly set the upper bound to be d4f$(\beta)/2$), the first J vectors in the `Pump`[5] output lattice basis are almost HKZ reduced. For each tour, calculate the ratio $l_i''/\mathrm{Sim}(l_i'')$ for $i \in [0, d-1]$, see

[5] To obtain such HKZ reduced basis, we should set `pump/down_sieve` = `True` and delete the condition "(`pump.insert_left_bound <= kappa+down_stop`)" in the file "`pump.py`" in the path "`G6K-GPU-Tensor/g6k/algorithms`" to make sure the output projected basis close to an HKZ reduction.

input : rr, blocksize $\beta \in \{45, \cdots, d\}$, jump J and number of tours t.
output: A prediction for the logarithms of the Gram-Schmidt norms $l'_i = \ln(\|\mathbf{b}'^*_i\|)$ after t tours PnjBKZ-β reduction with jump is J.

Function `PnjBKZSim(rr, β, J, t)`**:**

- **for** $i \leftarrow 0$ **to** 44 **do**
  - $r_i \leftarrow$ average $\ln(\|\mathbf{b}^*_i\|)$ of a HKZ reduced random unit-volume 45-dimensional lattice;
- **for** $i \leftarrow 45$ **to** β **do**
  - $c_i \leftarrow \ln\left(\mathrm{V}_i(1)^{-1/i}\right) = \ln\left(\frac{\Gamma(i/2+1)^{1/i}}{\pi^{1/2}}\right)$;
- **for** $j \leftarrow 0$ **to** $t-1$ **do**
  - flag $\leftarrow$ true; //flag to store whether $L_{[k,d]}$ has changed
  - **for** $k \leftarrow 0$ **to** $d-\beta-1$ **do**
    - $\beta' \leftarrow \min(\beta, d-k)$; //Dimension of local block
    - $h \leftarrow \min(k - (k \bmod \mathrm{J}) + \beta - 1, d-1)$;
    - $\ln(\mathrm{V}) \leftarrow \sum_{i=0}^{h} l_i - \sum_{i=0}^{k-1} l'_i$; //Let $\sum_{i=0}^{-1} l'_i = 0$
    - **if** flag = True **then**
      - **if** $\ln(\mathrm{V})/(\beta' - (k \bmod \mathrm{J})) + c_{\beta'-(k \bmod \mathrm{J})} < l_k$ **then**
        - $l'_k \leftarrow \ln(\mathrm{V})/(\beta' - (k \bmod \mathrm{J})) + c_{\beta'-(k \bmod \mathrm{J})}$;
        - flag $\leftarrow$ False;
    - **else**
      - $l'_k \leftarrow \ln(\mathrm{V})/(\beta' - (k \bmod \mathrm{J})) + c_{\beta'-(k \bmod \mathrm{J})}$;
  - **for** $k \leftarrow d-\beta$ **to** $d-46$ **do**
    - $\beta' \leftarrow d-k$; $h \leftarrow d-1$; $\ln(\mathrm{V}) \leftarrow \sum_{i=0}^{h} l_i - \sum_{i=0}^{k-1} l'_i$;
    - **if** flag = True **then**
      - **if** $\ln(\mathrm{V})/\beta' + c_{\beta'} < l_k$ **then**
        - $l'_k \leftarrow \ln(\mathrm{V})/\beta' + c_{\beta'}$; flag $\leftarrow$ false;
    - **else**
      - $l'_k \leftarrow \ln(\mathrm{V})/\beta' + c_{\beta'}$;
  - $\ln(\mathrm{V}) \leftarrow \sum_{i=0}^{h} l_i - \sum_{i=0}^{k-1} l'_i$;
  - **for** $k \leftarrow d-45$ **to** $d-1$ **do**
    - $l'_k \leftarrow \frac{\ln(\mathrm{V})}{45} + r_{k+45-d}$;
  - **for** $k \leftarrow 0$ **to** $d-1$ **do**
    - $l_k \leftarrow l'_k$;
- **return** $l_0, \cdots, l_{d-1}$;

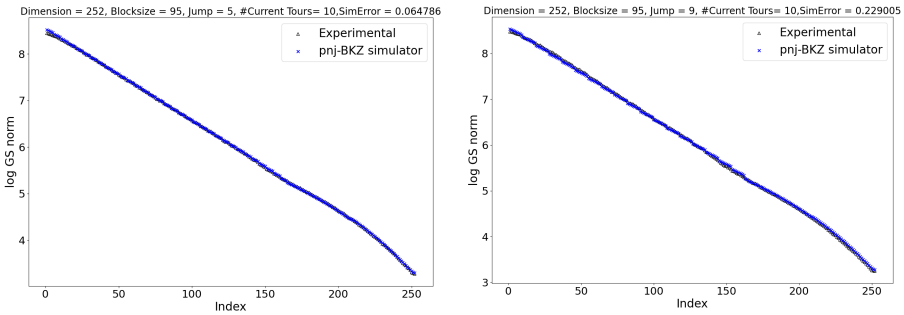
Algorithm 2: PnjBKZ Simulator

Fig. 3. Here l_i'' is the average logarithms of these Gram-Schmidt vector lengths obtained from 20 experiments, and $\mathrm{Sim}(l_i'')$ is the simulated logarithm of lengths of Gram-Schmidt vector which are calculated by Eq. (1).

We calculate the $\mathrm{Sim}(l_i'')$ strictly according to the property of the HKZ reduction basis and Heuristic 1. Therefore, in addition to being one of the criteria for measuring the accuracy of the PnjBKZ simulator, this ratio $l_i''/\mathrm{Sim}(l_i'')$ can also be used as a criterion for judging whether during the reduction of PnjBKZ, when the i-th `Pump` is called, the basis of $\mathcal{L}_{\pi[i:i+\beta]}$ reduced by Pump (Turn on sieving during the Pump-down stage) satisfies the HKZ reduction property[5].

However, it can be seen from Fig. 3 that for β=95, when $\mathsf{jump} \leq \mathrm{d4f}(\beta)/2 \approx 9$, even the tours increase to 10, the ratios $l_i''/\mathrm{Sim}(l_i'')$ are all between 0.975 and 1.025, indicating that the PnjBKZ simulator using Eq. (1) as the approximate estimate of the actual value l_i'' can already reflect how the average of the norms of Gram-Schmidt vectors change during each tour's reduction of PnjBKZ-(β, J).

Besides, when simulating the length value of Gram-Schmidt vector, we note that even for $i \equiv 1(\mathrm{mod}J)$, the index i of GH $\left(\mathcal{L}''_{\pi[i:i+\beta-(i \bmod J)]}\right)$ in our simulator is the same as that of GH $\left(\mathcal{L}'_{\pi[i:i+\beta]}\right)$ in the BKZ-2.0 simulator, the simulated volumes of projected sublattice $\mathcal{L}'_{\pi[i:i+\beta]}$ and $\mathcal{L}''_{\pi[i:i+\beta]}$ are different. The reason is that in BKZ 2.0 simulator [10] $\mathrm{Vol}(\mathcal{L}'_{\pi[i:i+\beta]}) = \prod_{j=0}^{i+\beta-1}\|\mathbf{b}_j^*\| / \prod_{j=0}^{i-1}\|\mathbf{b}_j^{*'}\|$ and it calculates $\|\mathbf{b}_j^{*'}\|$ by $\|\mathbf{b}_j^{*'}\| := \mathrm{GH}\left(\mathcal{L}'_{\pi[j:j+\beta]}\right)$, while in PnjBKZ simulator $\mathrm{Vol}(\mathcal{L}''_{\pi[i:i+\beta]}) = \prod_{j=1}^{i+\beta-1}\|\mathbf{b}_j^*\| / \prod_{j=1}^{i-1}\|\mathbf{b}_j^{*''}\|$ and $\|\mathbf{b}_j^{*''}\|$ obtained from Eq. (1). We give a detailed algorithm description of the PnjBKZ simulator in the Alg. 2.



(a) $\beta = 95, jump = 5, \sharp\mathrm{tours} = 10$ (b) $\beta = 95, jump = 9, \sharp\mathrm{tours} = 10$

Fig. 4. Prediction effect of PnjBKZ simulator. To verify the effectiveness of our PnjBKZ simulator, we perform the experiments by reducing the lattice basis of LWE Challenge ($n = 70, \alpha = 0.005$) by PnjBKZ with reduction parameter: blocksize $\beta = 95$, jump size $J \in [1, \cdots, 12]$, $\sharp\mathrm{tours} \in [1, \cdots, 10]$. We test 20 times for each reduction parameters.

Performance of PnjBKZ simulator We give an experiment to verify the effectiveness of our PnjBKZ simulator in this part.

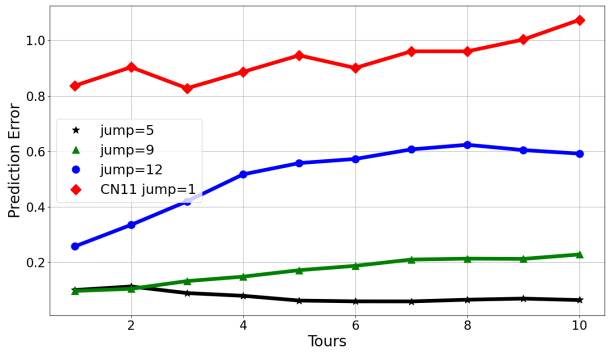


Fig. 5. Simulation error of PnjBKZ simulator with different jump values. For each reduction parameter tested by 20 experiments to get the average length of GS vectors.

To measure the accuracy of PnjBKZ simulator, SimError is calculated as following the equation: $\text{SimError}(\sharp\text{tours}) = \sum_{i=0}^{d-1} \left(\|\mathbf{b}_i^*\|_{(\sharp\text{tours})} - \text{Sim}(\|\mathbf{b}_i^*\|)_{(\sharp\text{tours})}\right)^2$, where $\sharp$tours represents the number of current tours and $\text{Sim}(\|\mathbf{b}_i^*\|)_{(\sharp\text{tours})}$ are the lengths of Gram-Schmidt vectors predicted by PnjBKZ simulator with $\sharp$tours.

As can be seen in the Fig. 4, PnjBKZ simulator can well predict the behavior of the PnjBKZ algorithm for β=95 when jump<10 since the corresponding SimError($\sharp$tours) is small. Fig. 5 shows that as the jump value increases, the prediction error increases, and the accuracy of PnjBKZ simulator decreases. However, β=95, the prediction error SimError($\sharp$tours) is still within 1 especially when the jump<13 and the SimError($\sharp$tours) of PnjBKZ simulator is not bigger than that of BKZ 2.0 simulator. Therefore the PnjBKZ simulator can predict how the average norms of Gram-Schmidt vectors change during each tour reduction of PnjBKZ-(β, J). See Appendix B for more details about the prediction effect of the simulator under different jump values and different tours.

4.2 Pump Cost Model and Dimension Estimation of the Last Pump

Pump Cost Model We set T_{Pump} as the time cost of a `Pump` since the main cost of a `Pump` can be regard as a $(\beta - f)$-dimensional progressive sieve, i.e.

$$T_{\text{Pump}}(\beta) = \sum_{j=\beta_0}^{\beta-f} T_{\text{sieve}}(j) \leq 2^{c\beta_0} \cdot \frac{2^{c(\beta-f+1)+o(\beta-f+1)}}{1-2^c} \approx 2^{c(\beta-f)+c_1}, \tag{2}$$

where β_0 is the dimension of initial sieving in `Pump` (In G6K β_0 set as 30, and in G6K-GPU, it set as 50), c and c_1 are the coefficients of sieve cost related to sieve algorithm, $T_{\text{sieve}}(j)$ is the sieve cost in dimension j with d4f value $f = \text{d4f}(j)$. More detail of our practical `Pump` cost model can be seen in the Appendix C.

Dimension Estimation of the Last Pump Except for the PnjBKZ simulator, to solve uSVP$_\gamma$ with high success probability, we need to determine the dimension of sieving in the last `Pump`. In this part, we propose a new dimension of sieving estimation of `Pump` named ProSieveDimEst (Alg. 3) to calculate the expected sieving dimension needed by a `Pump` to solve the uSVP$_\gamma$.

Specifically, $F(\star, \mathcal{D})$ as the input value of Alg. 3 is a distribution function, which describes the relationship between the length of the target vector projected on the projected sub-lattice and its probability. The idea, which considers the length of the projected target vector on the sub-lattice as a random variable rather than an expected value, was firstly proposed in [33]. Let $\star = \beta$, β represents the dimension β of projected lattice $\mathcal{L}_{\pi[d-\beta:d]}$ and $\mathcal{D}$ is the distribution of the length $\|\mathbf{t}\|$ of the projected target vector $\mathbf{t} \in \mathcal{L}_{\pi[d-\beta:d]}$.

It is efficient to calculate the probability in line 3 of the Alg. 3, because the dimension of the projected lattice d_{svp} will determine the upper limit of the integration of the probability density function, and the corresponding probability value can be easily obtained by doing the integration operation. Alg. 3 will evaluate the maximal sieve dimension d_{svp} for calling `Pump` within a high probability(>0.999) to find the target vector of uSVP$_\gamma$, then give a cost estimation.

Then we briefly explain the calculation of $\text{PSC}(d_{\text{svp}})$ in Alg. 3. Let $T_{\text{Pump}}(\beta) = \sum_{j=\beta_0}^{\beta} T_{\text{sieve}}(j)$ as Eq. (2). Since `Pump` contains the progressive sieve process, we should consider the failure/success probability during the process. A β-dimension progressive sieve runs sieve-i from some fixed β_0 to β. We set the success probability of β-dimension progressive sieve as $P_{\text{suc}}(\beta)$, then $P_{\text{suc}}(\beta) - P_{\text{suc}}(\beta-1)$ represents the success probability of the event E_β that one finds the target vector until the dimension of sieving exactly equals to β during the process of running the β-progressive sieve. The expected time cost of event E_β is $T_{\text{Pump}}(\beta) \cdot (P_{\text{suc}}(\beta) - P_{\text{suc}}(\beta-1))$. Iterate β from β_0 to d_{svp}, the expected `Pump` cost: $\text{PSC}(d_{\text{svp}}) = \sum_{\beta=\beta_0}^{d_{\text{svp}}} [T_{\text{Pump}}(\beta) \cdot \Pr(\text{E}_\beta)]$ calculated by Eq. (3)

$$\text{PSC}(d_{\text{svp}}) = \sum_{\beta=\beta_0}^{d_{\text{svp}}} \left[T_{\text{Pump}}(\beta) \cdot (P_{\text{suc}}(\beta) - P_{\text{suc}}(\beta-1))\right]. \tag{3}$$

Here $P_{\text{suc}}(d_{\text{svp}}) \geq 0.999$. More theoretical analysis about the calculation of $\text{PSC}(d_{\text{svp}})$: the success probability of solving the uSVP$_\gamma$ by a `Pump` with d_{svp}-dimension progressive sieve, see Eq. (11) in Appendix E.1, and for the corresponding verification experiments, see Appendix E.2.

In the case of solving standard form LWE, the fastest solving algorithm to it is the primal attack. The primal attack solves LWE problem by transforming it to uSVP$_\gamma$ Problem and then call a uSVP$_\gamma$ solver to solve it. Its target vector $\mathbf{t} = (\mathbf{e}, \pm 1)$ is a combination of ± 1 and the noise vector $\mathbf{e} \in \mathbb{Z}_q^m$ from a discrete Gaussian distribution D_σ with standard deviation σ.

The estimation of d_{svp} in the default G6K uses the estimation proposed in [13]. It computes the expected norm of the projected target vector $\|\pi_{d-d_{\text{svp}}}(\mathbf{t})\| \approx \sigma\sqrt{d_{\text{svp}}}$. It declared that it satisfies the condition $\sigma\sqrt{d_{\text{svp}}} \approx \|\pi_{d-d_{\text{svp}}}(\mathbf{t})\| \leq \text{GH}(\mathbf{B}_{\pi[d-d_{\text{svp}}]})$, then the projected shortest vector may be in the projected sub-

lattice. It outputs the minimal value d_{svp} such that the inequality $\sigma\sqrt{d_{\text{svp}}} \leq \text{GH}(\mathbf{B}_{\pi[d-d_{\text{svp}}]})$ holds and take it as the upper bound of sieving in the `Pump`. Since norm of the target vector maybe larger than its expected value in some cases, estimating the upper bound of `Pump` by the expected value is over-optimistic.

$\|\mathbf{e}\|^2$ is a randomly positive variable following chi-squared distribution (Definition 2) rather than a fixed value. It is more reasonable to consider a high success probability (≥ 0.999) for recovering the target vector with a suitable sieving dimension by setting $F(\beta, \mathcal{D}) = \sigma^2\chi^2_\beta$ in Alg. 3.

input : rr, $F(\star, \mathcal{D})$;
output: d_{svp}, PSC;
Function `ProSieveDimEst(`rr, $F(\star, \mathcal{D})$`)`**:**
  for $d_{\text{svp}} \leftarrow d_{\text{start}}$ **to** d **do**
    $P_{\text{suc}}(d_{\text{svp}}) \leftarrow \Pr\left[x \leftarrow F(d_{\text{svp}}, \mathcal{D}) \middle| x \leq \left(\text{GH}(\text{rr}_{[d-d_{\text{svp}}:d]})\right)^2\right]$;
    if $P_{\text{suc}}(d_{\text{svp}}) \geq 0.999$ **then**
      return d_{svp}, PSC(d_{svp});

Algorithm 3: Dimension Estimation for Pump on solving uSVP$_\gamma$.

4.3 Estimator Stability Assumption

In this part, we give some heuristic assumptions on our simulator to support the strategy generation algorithms which will be discussed in detail in Sec. 5. Since we use only Gaussian Heuristic in our PnjBKZ simulator, we can see that our PnjBKZ simulator fits well for all lattices satisfying Gaussian Heuristic. Furthermore, since the theoretical time cost of sieving algorithm is related only to lattice dimension, we can also assume that our time cost model fits well for almost all lattices. Therefore, we give Heuristic 3.

Heuristic 3 *The practical time cost model (Appendix C) and PnjBKZ simulator (Alg.2) fit the actual time cost of the* `Pump` *algorithm and PnjBKZ algorithm and the reduction effect of a series of PnjBKZ-*(β, J) *for almost all lattices.*

The quality of the lattice basis is not always improved by using a fixed PnjBKZ-(β, J). Using a fixed PnjBKZ-(β, J) tour repeatedly for reduction can only gradually approach a certain degree of reduction basis, but the number of tours of calling PnjBKZ-(β, J) to obtain such a reduction basis is unclear. However, we can give a heuristic assumption on an upper bound of lattice quality such that PnjBKZ-(β, J) will not improve a lattice basis with this quality.

Heuristic 4 *Let* $\mathbf{B}$ *be a lattice basis after reduction of several PnjBKZ-*(β_i, J_i) *tours,* $J_i \leq \text{d4f}(\beta_i)/2$. *If* $\mathbf{B}$ *has same quality with a BKZ-*β *reduced basis (in the term of PSC), then the basis cannot be further improved by a PnjBKZ-*(β, J) *tour for any* $J \geq 1$.

Heuristic 4 is reasonable if we assume that a lattice basis after PnjBKZ-(β_i, J_i), $J_i \leq \mathrm{d4f}(\beta_i)/2$ satisfies GSA. Since a BKZ-β reduced basis also satisfies GSA, we can assume that the lengths of Gram-Schmidt basis vectors in $\mathbf{B}$ and a BKZ-β reduced basis are the same, thus behave the same in the PnjBKZ simulator. Since a BKZ-β reduced basis cannot be improved by a PnjBKZ-(β, J) tour, it also happens for $\mathbf{B}$.

5 Improved Progressive PnjBKZ: Blocksize and Jump Strategy Optimization

In this section, we describe our two blocksize and jump strategy selection algorithms in detail. The first is *blocksize and jump strategy selection algorithm based on ProBKZ* (BSSA), based on the blocksize selection strategy in *Improved Progressive BKZ* [12]. The second is a new algorithm called *Enumeration for Blocksize and jump Strategy* (EnumBS), through which we can get a blocksize and jump strategy with minimum simulated time cost. We give both formal proof and experimental results to show that our new strategy selection algorithm is better than the algorithm based on ProBKZ.

5.1 Blocksize and Jump Strategy Selection based on ProBKZ

The *blocksize and jump strategy selection algorithm based on ProBKZ* (BSSA, Fig.6) applies the *Shortest Path Algorithm* to strategy selection.

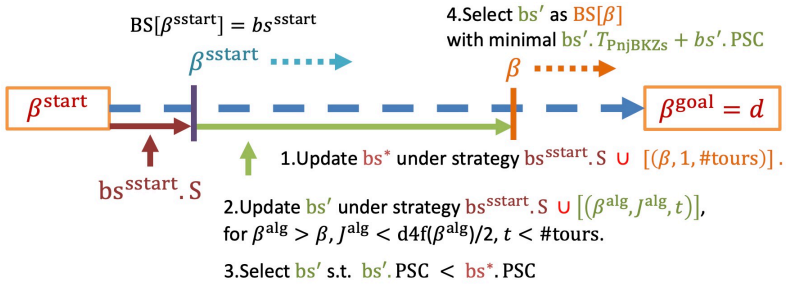


Fig. 6. BSSA Process.

BSSA starts with a fully BKZ-β^{start} reduced lattice basis. It firstly sets several middle nodes of β^{goal} as a measure of basis quality. For the edges between two nodes β_i and β_j, BSSA finds the optimal blocksize and jump $(\beta^{\mathrm{alg}}, J^{\mathrm{alg}})$ with tours t which reduces a BKZ-β_i reduced lattice basis into a BKZ-β_j reduced lattice basis, with minimum simulated time cost T_{PnjBKZ}. The tuple $(\beta^{\mathrm{alg}}, J^{\mathrm{alg}}, t, T_{\mathrm{PnjBKZ}})$ is stored in a list which the key is each edge in the graph.

For each node, we define a blocksize and jump strategy dictionary BS, in which the key is each middle β^{goal} node and the value is a tuple of $\mathsf{bs} = (\mathsf{rr}, \mathsf{S},$

T_{PnjBKZs}, PSC), where rr is the length of Gram-Schmidt vector which is fully BKZ-β^{goal} reduced, S means the blocksize and jump selection strategy which will improve the quality of lattice basis from fully BKZ-β^{start} reduced to fully BKZ-β^{goal} reduced, which is the combination of $(\beta^{\text{alg}}, J^{\text{alg}})$ tours stored on each edge in the shortest path from node β^{start} to β^{goal} with respect to the sum of simulated BKZ cost T_{PnjBKZs}, while the shortest path can be found using Dijkstra algorithm. PSC is one of the output from the `Pump` dimension estimation method (Alg.3), which means the estimated time cost for uSVP$_\gamma$ to be solved by processing `Pump` on the BKZ-β^{goal} reduced basis.

To find the efficient two-step solving strategy, we should consider the cost of the last `Pump`, i.e. PSC, after several PnjBKZ reductions. By setting different final β^{goal}, we can get different reduction strategy BS that improves the quality of lattice basis from β^{start} to β^{goal} and different sieving dimension of the last `Pump` corresponding to the different quality of the lattice that is fully β^{goal} reduced. Then we set multiple different final β^{goal} to choose the two-step solving strategy whose total time cost is minimum. Here, the total time cost includes the time cost of improving the quality of lattice by a series of PnjBKZ-$(\beta, J) \in \mathsf{S}$ and the time cost of final `Pump`. See Alg. 11 for more details about BSSA.

5.2 Blocksize and Jump Strategy Enumeration Algorithm

The advantage of BSSA is that the algorithm runs in polynomial time. However, there are also some disadvantages, we list two of them.

First, the reduction effect of a BSSA strategy is not estimated accurately. Each intermediate result of lattice basis is fitted into one of the nodes, which is a BKZ-β reduced basis for some β, but the acutual basis may have higher quality than the node. Thus a BSSA strategy may overestimate the time cost, which leads to the non-optimization of the output strategy.

Second, BSSA also misses many potential optimized strategies. For example, BSSA only considers reducing a BKZ-β reduced basis into a BKZ-$(\beta+1)$ reduced basis with BKZ tours of fixed blocksize. However, the optimal reduction between the two nodes might be progressive, and cannot be found by BSSA.

The problems can be solved by using a blocksize and jump strategy enumeration algorithm to enumerate all possible blocksize and jump strategies, so that we can always find the optimized strategy among them.

However, an enumeration algorithm is inefficient, and cannot be used in practice since it costs an exponential time overhead. We improve the algorithm using a branch and bound method. To design an algorithm on pruning the strategy in the branch and bound, we need another heuristic assumption.

Heuristic 5 *Given two sufficiently reduced lattice bases* $\mathbf{B}, \mathbf{B}'$ *with lengths* $\mathsf{rr}, \mathsf{rr}'$. *If* $\text{PSC}(\mathsf{rr}) \geq \text{PSC}(\mathsf{rr}')$, *then* $\text{PSC}(\text{PnjBKZSim}(\mathsf{rr}, \beta, J, 1)) \geq \text{PSC}(\text{PnjBKZSim}(\mathsf{rr}', \beta, J, 1))$ *also holds for any* β *and* $J \leq \text{d4f}(\beta)/2$.

This assumption holds trivially if we assume that $\mathbf{B}$ and $\mathbf{B}'$ satisfy GSA. Note that all heuristic assumptions we used in this work is weaker than GSA

(Definition 2). Since GSA is always considered as a reliable assumption and widely used in many lattice solving algorithms and estimators, it is sufficient to say that our strategy selection algorithm can output a blocksize and jump strategy with expected minimal cost in solving uSVP_γ for any randomly generated lattice instance.

input : rr_0, $F(\star, \mathcal{D})$, $J_{\max}(\star) \leftarrow \text{d4f}(\star)/2$;
output: $T_{\min}$, $\mathsf{S}_{\min}$;
Function `EnumBS`(rr_0, $F(\star, \mathcal{D})$, $J_{\max}(\star) \leftarrow \text{d4f}(\star)/2$)**:**
  $k \leftarrow 1$; $d \leftarrow \text{len}(\mathsf{rr}_0)$; $\_, \text{PSC}^{(0)} \leftarrow$ `ProSieveDimEst`(rr_0, $F(\star, \mathcal{D})$);
  $\mathsf{bs}^{(0)} \leftarrow (\mathsf{rr}_0, [\,], 0, \text{PSC}^{(0)})$; $\mathsf{BS} \leftarrow \{\mathsf{bs}^{(0)}\}$;
  while $k \leq \sharp\mathsf{BS}$ **do**
    $\mathsf{bs} \leftarrow \mathsf{BS}\,[k]$; $(\beta, J) \leftarrow$ the last element of $\mathsf{BS}[k].\mathsf{S}$;
    $(\beta, J) \leftarrow$ next (β, J) s.t. $\text{PSC}($`PnjBKZSim` $(\mathsf{bs.rr}, \beta, J, 1)) < \mathsf{bs.PSC}$;
    while (β, J) is not None **do**
      $\mathsf{bs}^*.\mathsf{S} \leftarrow \mathsf{S} \cup [(\beta, J)]$, update bs^* under $\mathsf{bs}^*.\mathsf{S}$;
      $\mathsf{BS} \leftarrow \mathsf{BS} \cup \{\mathsf{bs}^*\}$;
      if $\exists\ \mathsf{bs} \in \mathsf{BS}$ s.t. $\mathsf{bs}^*.\text{PSC} \geq \mathsf{bs.PSC}$ **and** $\mathsf{bs}^*.T_{\text{PnjBKZs}} \geq \mathsf{bs}.T_{\text{PnjBKZs}}$ **then**
        $\mathsf{BS} \leftarrow \mathsf{BS} \setminus \{\mathsf{bs}^*\}$;
      else
        for $\forall \mathsf{bs} \in \mathsf{BS}$ s.t. $\mathsf{bs}^*.\text{PSC} \leq \mathsf{bs.PSC}$ **and** $\mathsf{bs}^*.T_{\text{PnjBKZs}} \leq \mathsf{bs}.T_{\text{PnjBKZs}}$ **do**
          $\mathsf{BS} \leftarrow \mathsf{BS} \setminus \{\mathsf{bs}\}$;
      $(\beta, J) \leftarrow$ next (β, J) s.t. $\text{PSC}($`PnjBKZSim` $(\mathsf{bs.rr}, \beta, J, 1)) < \mathsf{bs.PSC}$;
      $k \leftarrow k + 1$;
  $\mathsf{bs}_{\min} \leftarrow \min_{\mathsf{bs}.T_{\text{PnjBKZs}} + \mathsf{bs.PSC}} \mathsf{BS}$;
  return $T_{\min} \leftarrow \mathsf{bs}_{\min}.T_{\text{PnjBKZs}} + \mathsf{bs}_{\min}.\text{PSC}$, $\mathsf{S}_{\min} \leftarrow \mathsf{bs}_{\min}.\mathsf{S}$;

Algorithm 4: EnumBS

The detailed description of EnumBS with pruning shown in Alg. 4. In EnumBS, we use BS to store the information of each reduction strategy which might be the optimal strategy or will become the optimal strategy after adding more (β, J) nodes. BS is a list and each element bs in the BS is a tuple of values $\mathsf{bs} = (\mathsf{rr}, \mathsf{S}, T_{\text{PnjBKZs}}, \text{PSC})$. It is important to note that each bs in BS should be in order that increases by its T_{PnjBKZs} value and decreases by its PSC value. In bs, S is a list storing the blocksize and jump strategy used for calling PnjBKZ, T_{PnjBKZs} is the time cost for calling such a series of PnjBKZ, rr stores the current simulated gs-lengths after simulatedly calling PnjBKZ following strategy S by PnjBKZ simulator (Alg.2). PSC output from the `Pump` dimension estimation method (Alg.3), it estimates the expected time cost for last `Pump`. Each element in S is a tuple (β, J), where β is the blocksize value of PnjBKZ and J is the

jump value of PnjBKZ. For the sake of narrative simplicity, we will use $\mathsf{bs}.\star$ to denote each element in bs, e.g. $\mathsf{bs}.\mathsf{S}$. Let $\sharp\mathsf{S}$ and $\sharp\mathsf{BS}$ be the length of S and BS.

At the start of EnumBS, there is only one tuple $\mathsf{bs}^{(0)}$ in BS, where $\mathsf{bs}^{(0)}.\mathsf{S} = [\,]$ denotes a no PnjBKZ blocksize and jump strategy with a pure `Pump` sieve. The total cost of $\mathsf{bs}^{(0)}$ is the `Pump` cost. Then, to generate more strategies and try to find the optimal strategy, we can regard $\mathsf{bs}^{(0)}$ as the root node and expand the strategy list from $\mathsf{bs}^{(0)}$ using a breadth-first search.

For a node bs in the tree, each of its children bs^* satisfies that $\mathsf{bs}^*.\mathsf{S} = \mathsf{bs}.\mathsf{S} \cup [(\beta, J)]$, where $\mathrm{PSC}(\mathrm{PnjBKZSim}(\mathsf{bs}.\mathsf{rr}, \beta, J, 1)) < \mathsf{bs}.\mathrm{PSC}$, which means that a (β, J) tour can further improve the basis quality. Assuming Heuristic 4, we can begin the search from $\beta_0 + 1$, where $\mathsf{bs}.\mathsf{rr}$ has equal or higher quality than a BKZ-β_0 reduced basis.

Considering each child strategy $\mathsf{bs}^*.\mathsf{S}$ of $\mathsf{bs}.\mathsf{S}$ for all possible (β, J), compute the other values in bs^*, i.e. $\mathsf{bs}^*.T_{\mathrm{PnjBKZs}}$, $\mathsf{bs}^*.\mathsf{rr}$and $\mathsf{bs}^*.\mathrm{PSC}$. When we try to add a bs^* into BS, we should first determine whether it exists a $\mathsf{bs}' \in \mathsf{BS}$ so that $\mathsf{bs}^*.\mathrm{PSC} \geq \mathsf{bs}'.\mathrm{PSC}$ and $\mathsf{bs}^*.T_{\mathrm{PnjBKZs}} \geq \mathsf{bs}'.T_{\mathrm{PnjBKZs}}$. If so, we cannot add such bs^* into BS, because the child strategies generated by bs^*(including bs^* itself) won't have a shorter time overhead than which generated by the corresponding bs'. If not, then we should first add bs^* and then delete the bad strategy in BS whose PSC value and T_{PnjBKZs} value are both larger than bs^*. Iterate each $\mathsf{BS}[k]$ and its child nodes sequentially, we will end up with a BS containing the block size strategy with minimum simulated time cost. Finally, we search through BS and return the best strategy in the end.

In addition, Theorem 1 proves that EnumBS can find the blocksize and jump strategy with minimal simulated cost to solve uSVP_γ in two-step mode.

Theorem 1. *Let $\mathcal{S}$ be the set of all sequences consist of $(\beta, J \leq \mathrm{d4f}(\beta)/2)$, $\mathcal{S}$ is the set of all possible blocksize and jump strategies. If Heuristic 5 holds, the algorithm EnumBS always returns the blocksize strategy in $\mathcal{S}$ with minimum simulated time cost.*

Proof. Let rr_0 be the input basis quality, Suppose that the strategy in $\mathcal{S}$ with minimum simulated cost is $\mathsf{S} = [(\beta_1, J_1), ..., (\beta_k, J_k)]$. We write the sub-strategy $[(\beta_1, J_1), ..., (\beta_i, J_i)]$ of S, $i \leq k$ as S_i.

We can see that $\mathrm{PSC}(\mathrm{PnjBKZSim}(\mathsf{rr}_0, \mathsf{S}_{i-1})) \geq \mathrm{PSC}(\mathrm{PnjBKZSim}(\mathsf{rr}_0, \mathsf{S}_i))$ for all $i \leq k$, otherwise removing (β_i, J_i) from S can get a better strategy by Heuristic 5. From the description of EnumBS, either S is inside the final strategy set BS, or there is a sub-strategy S_i such that S_i is removed from BS (then S won't appear in BS anymore). Since S has minimum time cost among all strategies in BS, S must be the final output strategy and meets the first case. Now we show that the second case cannot occur.

If S_i is removed from BS, then there must be another strategy S' such that $\mathrm{PSC}(\mathrm{PnjBKZSim}(\mathsf{rr}_0, \mathsf{S}_i)) \geq \mathrm{PSC}(\mathrm{PnjBKZSim}(\mathsf{rr}_0, \mathsf{S}'))$, and the PnjBKZ time cost $\mathsf{S}_i.T_{\mathrm{PnjBKZs}} \geq \mathsf{S}'.T_{\mathrm{PnjBKZs}}$. If we append the sequence $(\beta_{i+1}, J_{i+1}), ..., (\beta_k, J_k)$ into S' and get a new strategy S^*, it infers that $\mathsf{S}.T_{\mathrm{PnjBKZs}} \geq \mathsf{S}^*.T_{\mathrm{PnjBKZs}}$.

By Heuristic 5, $\mathrm{PSC}(\mathrm{PnjBKZSim}(\mathsf{rr}_0, \mathsf{S})) \geq \mathrm{PSC}(\mathrm{PnjBKZSim}(\mathsf{rr}_0, \mathsf{S}^*))$. Now we have that S^* has expected smaller time cost than S, which makes a contradiction. Thus we finish the proof. □

6 Apply ProPnjBKZ to LWE

In this section, we mainly show the performance of our ProPnjBKZ to solve LWE in the two-step mode. Specifically, in Sec. 6.1, we give the optimized blocksize and jump strategy for solving the TU Darmstadt LWE Challenge. In Sec. 6.2, we apply our ProPnjBKZ to TU Darmstadt LWE Challenge and give a comparison among different LWE-solving modes by experiments. In Sec. 6.3, wwe show the new TU Darmstadt LWE Challenges recording we have broken. In Sec. 6.4, we give a new security estimation of LWE in NIST schemes [28] based on the two-step solving mode and EnumBS.

6.1 ProPnjBKZ for Solving TU Darmstadt LWE Challenge

We use a machine with Intel Xeon 5128 16c 32@2.3GHz, 1.48T RAM and NVIDIA Geforce RTX 3090 * 2 to construct a practical time-cost model by experiments, see more detail in the Appendix C. Based on this practical time-cost model we give a comparison of different LWE-solving algorithms.

Table 3. Blocksize and Jump strategy generated by EnumBS.

(n, α)	Strategy (β, jump)	EnumBSGen/s
(40,0.025)	[(89,9),(114,10)]	100
(40,0.030)	[(73,8),(89,9),(117,10),(119,10)]	460
(45,0.020)	[(73,8),(90,9),(117,10)]	220
(50,0.015)	[(73,8),(90,9),(114,10)]	280

We use EnumBS (Alg. 4) with the practical cost model to select the blocksize and jump strategy for some instances of TU Darmstadt LWE Challenges, we list the selected strategies in Table 3. Besides, Table 3 shows that the time cost of generating the reduction strategy by EnumBS is acceptable. Also, we upload the open source code for blocksize and jump strategy generation on any LWE instances[1]. We solved the TU Darmstadt LWE Challenge instances with $(n, \alpha) \in \{(40, 0.035), (40, 0.040), (50, 0.025), (55, 0.020), (90, 0.005)\}$ successfully by the selected strategies in Appendix F.

6.2 Experimental Comparison of LWE-solving Algorithms

In this part, we test the walltime of different LWE solving algorithms through experiments to prove that our algorithm is indeed improved compared with the default LWE solving algorithm in G6K.

The default LWE solving algorithm in G6K is the script `lwe_challenge.py` in the implementation of G6K's GPU version [24], we have discussed it in Sec. 3.1. Besides, for more detail about the default LWE solving algorithm in G6K's GPU version[6]. Table 4 gives the experimental result of different LWE-solving algorithms. The columns of "G6K" is the experimental time/memory cost of default strategy in G6K. The columns of "EnumBS"("BSSA") give the experimental time/memory cost using the strategy generated by EnumBS(Alg. 4) (BSSA(Alg. 11)). From the result of Table 4, we can see that using the strategy selected by EnumBS (BSSA) significantly decreased the walltime cost by about 7.6∼12.9 (5.5∼11.7) times compared to the default LWE solving strategy in G6K. One can refer to the full vision of Table 4 in the folder `lwechal-test`. It can also be reproduced by running the test code `implement_low_dim.sh`[2].

Table 4. Experimental Result of Different LWE-solving Algorithms[§].

(n, α)	Walltime/s			Memory/GB			TAR	
	G6K	BSSA	EnumBS	G6K	BSSA	EnumBS	BSSA	EnumBS
$(40, 0.025)$	11864	2153	1571	4.5	27.2	3.2	5.5	7.6
$(40, 0.030)$	70671	6020	5495	10.1	35.1	46.8	11.7	12.9
$(45, 0.020)$	22400	3267	2569	1.7	30.9	5.0	6.9	8.7
$(50, 0.015)$	20185	2414	2390	11.2	11.2	3.4	8.4	8.4

§ Here EnumBS(BSSA) represents the two-step LWE solving algorithm whose reduction strategy is generated by Alg. 4(Alg. 11) in a machine of 32 threads(Intel Xeon 5128 16c 32@2.3GHz), 1.48T RAM and 2 gpus(NVIDIA Geforce RTX 3090). "TAR" means the Walltime Acceleration Ratio computed in Walltime(G6K)/Walltime(BSSA) or Walltime(G6K)/Walltime(EnumBS).

Meanwhile, to show the accuracy of our optimized blocksize and jump selected strategy, we compare the quality of lattice basis and walltime in each middle node predicted by the selected blocksize and jump strategy with that of actual experiments. Table 5 illustrates that both the quality of the actual lattice basis and the actual walltime of each tour of PnjBKZ-(β, J) are close to our prediction.[7] It also proves that Heuristic 3 established in an experimental way.

6.3 New LWE Records

TU Darmstadt LWE Challenge website presents Challenges for testing the efficiency of solving LWE which helps to estimate the hardness of LWE in practice.

By our new algorithm, i.e. ProPnjBKZ, we have solved the LWE instances $(n, \alpha) \in \{(40, 0.035), (90, 0.005), (50, 0.025), (55, 0.020), (40, 0.040)\}$ in TU Darmstadt LWE Challenge website. (We solved the LWE instance $(n, \alpha) = (80, 0.005)$

[6] https://github.com/WvanWoerden/G6K-GPU-Tensor/blob/main/lwe_challenge.py

[7] The data in Table 5 is extracted from a test in Table 4 for comparing the quality and walltime between our simulations and actual experiments.

Table 5. Quality(walltime T/s) during reduction of LWE $(n, \alpha) = (40, 0.030)$.

(β, J)	Simulation		Practical	
	Slope	log(T)	Slope	log(T)
(73,8)	−0.0547	8.0	−0.0520	8.2
(89,9)	−0.0469	8.7	−0.0455	8.8
(117,10)	−0.0400	10.0	−0.0396	10.5
(119,10)	−0.0381	10.4	−0.0381	10.8

Table 6. Actual running time, RAM cost and simulation cost.

(n,α)	Machine	T (h)	RAM (GB)
(40,0.035)	B	233	184
(50,0.025)	A	592	184
(55,0.020)	A	611	890
(90,0.005)	B	370	332
(40,0.040)	A	683	1120

earlier.)[8] Specifically we denoted a service with AMD EPYC™ 7002 Series 128@2.6GHz, NVIDIA RTX 3090 * 8, 1.5T RAM as Machine A, and denoted a service with AMD EPYC™ 7002 Series 64@2.6GHz, a100 *4, 512 GB RAM as Machine B. Then we listed the walltime and RAM cost in solving the above LWE Challenges in Table 6. The unit of time in Table 6 is hour and the unit of RAM is GB.

6.4 Security Estimation for NIST schemes

In this part, we estimate the security bit of LWE based NIST schemes [28] under consideration of the influence of the two-step mode and the blocksize and jump strategy selection. Our new concrete hardness estimation of LWE tries to answer Question 7 in Section 5.3 of [2] and narrows the security estimation error interval. For more details about the construction of our new concrete hardness estimator of two-step mode of solving LWE, we refer to Appendix E and Appendix G for more estimation results. Now our evaluation code is available at open source[1].

We compare our estimation with Leaky-LWE-Estimator [42] under the same gate-count algorithm [30] and same heuristic assumptions. Under the RAM model, i.e, it assumes that access into even exponentially large memory is free, the estimated security bit of LWE in NIST schemes [28] can be reduced by $1.9 \sim 4.2$ bit compared to the current estimation generated by Leaky-LWE-Estimator[9] in [42]. See Table 7 for details. Here G and B in Table 7 respectively represent the total number of logic circuits for these LWE instances in NIST schemes [28] being solved and the maximal memory needed for solving these LWE instances, that both are calculated by Gate-count algorithm [30].

Without considering the RAM model, Two-step mode of using larger `Pump` dimension will indeed lead to an extra cost of accessing exponentially large memory, which will somewhat offset the above-claimed decreasing of security hardness. But since the time cost (Number of gates G) is far larger than than the memory cost B, the impact of memory growth can be ignored with such a significant decreasing in time cost. Specifically, in Table 7 even though ΔB is negative (Two-step mode use bigger memory), $\Delta \log_2(\mathrm{G} + \mathrm{B})$ same as $\Delta \log_2 \mathrm{G}$ between 1.9∼4.2 bit is still positive. It means that the increase in memory will

[8] https://www.latticechallenge.org/lwe_challenge/challenge.php
[9] https://github.com/lducas/leaky-LWE-Estimator

partially offset the decrease in the number of gates. However, in general, the time cost is still decreasing even considering the extra increase in memory.

Table 7. Security Estimation results of different estimator for NIST schemes[♮].

	$\log_2$G/log2(gate)			$\log_2$B/log2(bit)			ΔG
	Previous	Two-Step	EnumBS	Previous	Two-Step	EnumBS	
Kyber512	151.5	148.6	147.4	93.8	99.1	98.1	4.1
Kyber768	215.1	212.1	210.9	138.5	144.0	143.2	4.2
Kyber1024	287.3	284.2	283.4	189.7	195.4	194.6	3.9
Dilithium-II	158.6	156.7	156.6	97.8	104.3	104.4	2.0
Dilithium-III	216.7	214.5	214.5	138.7	145.3	145.3	2.2
Dilithium-V	285.4	283.5	283.5	187.4	194.5	194.5	1.9

♮ The column "Previous" is the Security estimation in the statement of Kyber and Dilithium. "Two-step" is using a trial progressive BKZ+Pump in two step mode to estimate security. "EnumBS" is the estimated security by optimized strategy. ΔG is the difference between "Previous" and "EnumBS" under the RAM model. However, the list-decoding method proposed by MATZOV [32] can further decrease security bit and Albrecht *et al* adapted it to construct their estimator (https://github.com/malb/lattice-estimator). It helps us give the estimation in Appendix G.

Last but not least, although we only compare our security estimation with Leaky-LWE-Estimator [43] in this section, for other LWE estimators which consider only the BKZ-only mode, using our two-step mode along with the time-cost models in these estimators will also lead to better security estimation.

7 Conclusion and Future Work

In this paper, we propose the ProPnjBKZ, which combines PnjBKZ and `Pump` to solve the uSVP$_\gamma$ problem based on two new simulators (PnjBKZ simulator and `Pump` estimator). Experimental results show that our simulators can accurately predict the behavior of PnjBKZ even if jump>1. We design two new blocksize and jump strategy selection algorithms: BSSA and EnumBS, and demonstrate that the EnumBS strategy has minimal simulation time cost. Meanwhile, applying the blocksize and jump strategy generated from EnumBS to solve the LWE Challenge results in 7.6∼12.9 times improvement compared to default G6K mode and help us solve the TU Darmstadt LWE Challenges $(n, \alpha) \in \{(40, 0.035), (40, 0.040), (50, 0.025), (55, 0.020), (90, 0.005)\}$. In addition, using our new hardness estimator of LWE for security evaluation of NIST lattice-based schemes shows that our optimized blocksize and jump strategy selection and two-step mode will cause the security strength to drop by 1.9∼4.2 bit (8∼10.7 bit when the list-decoding method proposed by MATZOV is used).

We show some future works: A good insertion in `Pump` could decrease the time cost and increase the quality of basis, but the insert function in the `Pump` in

the G6K is heuristic. So we plan to study the insert function and try to design a more fittable function. In addition, we noticed that when using G6K to solve the high-dimension lattice challenge, the program often displays a saturation warning which we don't know what caused. However, it often results in longer time cost and we plan to solve this problem. Besides, we plan to give details of our new security estimation of the hardness of LWE in the NIST schemes by considering the influence of our optimized blocksize and jump selection and two-step mode strategy.

References

1. L. Ducas, T. L. Eike Kiltz, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, *Dilithium(Round 3)*. NIST PQC probject, 2020.
2. R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Kyber(Round 3)," p. 42, 2020.
3. O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM*, vol. 56, pp. 34:1–34:40, Sept. 2009.
4. V. Lyubashevsky, C. Peikert, and O. Regev, "On Ideal Lattices and Learning with Errors over Rings," in *Advances in Cryptology – EUROCRYPT 2010* (H. Gilbert, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 1–23, Springer, 2010.
5. S. Bai and S. D. Galbraith, "An Improved Compression Technique for Signatures Based on Learning with Errors," in *Topics in Cryptology – CT-RSA 2014* (J. Benaloh, ed.), (Cham), pp. 28–47, Springer International Publishing, 2014.
6. R. Kannan, "Improved algorithms for integer programming and related lattice problems," in *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, (New York, NY, USA), pp. 193–206, Association for Computing Machinery, Dec. 1983.
7. A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, pp. 515–534, Dec. 1982.
8. C. P. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," in *Fundamentals of Computation Theory* (L. Budach, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 68–85, Springer, 1991.
9. N. Gama, P. Q. Nguyen, and O. Regev, "Lattice Enumeration Using Extreme Pruning," in *Advances in Cryptology – EUROCRYPT 2010* (D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, and H. Gilbert, eds.), vol. 6110, pp. 257–278, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. Series Title: Lecture Notes in Computer Science.
10. Y. Chen and P. Q. Nguyen, "BKZ 2.0: Better Lattice Security Estimates," in *Advances in Cryptology – ASIACRYPT 2011* (D. H. Lee and X. Wang, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 1–20, Springer, 2011.
11. M. R. Albrecht, S. Bai, J. Li, and J. Rowell, "Lattice Reduction with Approximate Enumeration Oracles," in *Advances in Cryptology – CRYPTO 2021* (T. Malkin and C. Peikert, eds.), Lecture Notes in Computer Science, (Cham), pp. 732–759, Springer International Publishing, 2021.

12. Y. Aono, Y. Wang, T. Hayashi, and T. Takagi, "Improved Progressive BKZ Algorithms and Their Precise Cost Estimation by Sharp Simulator," in *Advances in Cryptology – EUROCRYPT 2016* (M. Fischlin and J.-S. Coron, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 789–819, Springer, 2016.
13. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum Key Exchange—A New Hope," pp. 327–343, 2016.
14. M. R. Albrecht, F. Göpfert, F. Virdia, and T. Wunderer, "Revisiting the expected cost of solving usvp and applications to lwe," in *Advances in Cryptology – ASIACRYPT 2017* (T. Takagi and T. Peyrin, eds.), (Cham), pp. 297–322, Springer International Publishing, 2017.
15. M. R. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. W. Postlethwaite, and M. Stevens, "The General Sieve Kernel and New Records in Lattice Reduction," in *Advances in Cryptology – EUROCRYPT 2019* (Y. Ishai and V. Rijmen, eds.), (Cham), pp. 717–746, Springer International Publishing, 2019.
16. D. Micciancio and P. Voulgaris, "Faster exponential time algorithms for the shortest vector problem," in *Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pp. 1468–1480, Society for Industrial and Applied Mathematics, Jan. 2010.
17. R. Fitzpatrick, C. Bischof, J. Buchmann, Ö. Dagdelen, F. Göpfert, A. Mariano, and B.-Y. Yang, "Tuning gausssieve for speed," in *Progress in Cryptology - LATINCRYPT 2014* (D. F. Aranha and A. Menezes, eds.), (Cham), pp. 288–305, Springer International Publishing, 2015.
18. P. Q. Nguyen and T. Vidick, "Sieve algorithms for the shortest vector problem are practical," *Journal of Mathematical Cryptology*, vol. 2, Jan. 2008.
19. G. Herold and E. Kirshanova, "Improved Algorithms for the Approximate k-List Problem in Euclidean Norm," in *Public-Key Cryptography – PKC 2017* (S. Fehr, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 16–40, Springer, 2017.
20. G. Herold, E. Kirshanova, and T. Laarhoven, "Speed-Ups and Time–Memory Trade-Offs for Tuple Lattice Sieving," in *Public-Key Cryptography – PKC 2018*, pp. 407–436, Springer, Cham, Mar. 2018.
21. A. Becker, N. Gama, and A. Joux, "Speeding up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search," 2015.
22. T. Laarhoven and A. Mariano, "Progressive Lattice Sieving," in *Post-Quantum Cryptography* (T. Lange and R. Steinwandt, eds.), (Cham), pp. 292–311, Springer International Publishing, 2018.
23. L. Ducas, "Shortest Vector from Lattice Sieving: A Few Dimensions for Free," in *Advances in Cryptology – EUROCRYPT 2018* (J. B. Nielsen and V. Rijmen, eds.), (Cham), pp. 125–145, Springer International Publishing, 2018.
24. L. Ducas, M. Stevens, and W. van Woerden, "Advanced Lattice Sieving on GPUs, with Tensor Cores," in *Advances in Cryptology – EUROCRYPT 2021* (A. Canteaut and F.-X. Standaert, eds.), Lecture Notes in Computer Science, (Cham), pp. 249–279, Springer International Publishing, 2021.
25. A. Becker, L. Ducas, N. Gama, and T. Laarhoven, "New directions in nearest neighbor searching with applications to lattice sieving," in *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '16, (USA), pp. 10–24, Society for Industrial and Applied Mathematics, Jan. 2016.
26. G. Hanrot, X. Pujol, and D. Stehlé, "Analyzing Blockwise Lattice Algorithms Using Dynamical Systems," in *Advances in Cryptology – CRYPTO 2011* (P. Rogaway, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 447–464, Springer, 2011.

27. S. Bai, D. Stehlé, and W. Wen, "Measuring, Simulating and Exploiting the Head Concavity Phenomenon in BKZ," in *Advances in Cryptology – ASIACRYPT 2018* (T. Peyrin and S. Galbraith, eds.), Lecture Notes in Computer Science, (Cham), pp. 369–404, Springer International Publishing, 2018.
28. I. T. L. C. S. R. CENTER, "Post-quantum cryptography pqc selected algorithms 2022." https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.
29. D. Dachman-Soled, L. Ducas, H. Gong, and M. Rossi, "Lwe with side information: Attacks and concrete security estimation," in *Advances in Cryptology – CRYPTO 2020* (D. Micciancio and T. Ristenpart, eds.), (Cham), pp. 329–358, Springer International Publishing, 2020.
30. M. R. Albrecht, V. Gheorghiu, E. W. Postlethwaite, and J. M. Schanck, "Estimating quantum speedups for lattice sieves," in *Advances in Cryptology – ASIACRYPT 2020* (S. Moriai and H. Wang, eds.), (Cham), pp. 583–613, Springer International Publishing, 2020.
31. M. R. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. W. Postlethwaite, and M. Stevens, "The General Sieve Kernel and New Records in Lattice Reduction," in *Advances in Cryptology – EUROCRYPT 2019* (Y. Ishai and V. Rijmen, eds.), vol. 11477, pp. 717–746, Cham: Springer International Publishing, 2019. Series Title: Lecture Notes in Computer Science.
32. MATZOV, "Report on the Security of LWE: Improved Dual Lattice Attack," Apr. 2022.
33. E. W. Postlethwaite and F. Virdia, "On the Success Probability of Solving Unique SVP via BKZ," in *Public-Key Cryptography – PKC 2021* (J. A. Garay, ed.), vol. 12710, pp. 68–98, Cham: Springer International Publishing, 2021. Series Title: Lecture Notes in Computer Science.
34. P.-Q. Chen, Yuanmi; Nguyen, *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*. PhD Thesis, 2013.
35. V. Lyubashevsky and D. Micciancio, "On Bounded Distance Decoding, Unique Shortest Vectors, and the Minimum Distance Problem," in *Advances in Cryptology - CRYPTO 2009* (S. Halevi, ed.), vol. 5677, pp. 577–594, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Series Title: Lecture Notes in Computer Science.
36. M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of Learning with Errors," *Journal of Mathematical Cryptology*, vol. 9, Jan. 2015.
37. C. Peikert, "A Decade of Lattice Cryptography," *Found. Trends Theor. Comput. Sci.*, vol. 10, pp. 283–424, Mar. 2016. Place: Hanover, MA, USA Publisher: Now Publishers Inc.
38. K. Xagawa, "Cryptography with Lattices," p. 244, 2010.
39. T. Laarhoven, "Sieving for Shortest Vectors in Lattices Using Angular Locality-Sensitive Hashing," in *Advances in Cryptology – CRYPTO 2015* (R. Gennaro and M. Robshaw, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 3–22, Springer, 2015.
40. A. Becker and T. Laarhoven, "Efficient (Ideal) Lattice Sieving Using Cross-Polytope LSH," in *Progress in Cryptology – AFRICACRYPT 2016* (D. Pointcheval, A. Nitaj, and T. Rachidi, eds.), (Cham), pp. 3–23, Springer International Publishing, 2016.
41. L. Babai, "On Lovász' lattice reduction and the nearest lattice point problem," *Combinatorica*, vol. 6, pp. 1–13, Mar. 1986.
42. D. Dachman-Soled, L. Ducas, H. Gong, and M. Rossi, "LWE with Side Information: Attacks and Concrete Security Estimation," in *Advances in Cryptology – CRYPTO*

2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II, (Berlin, Heidelberg), pp. 329–358, Springer-Verlag, Aug. 2020.
43. L. Ducas, "leaky-LWE-Estimator."
44. M. R. Albrecht, F. Göpfert, F. Virdia, and T. Wunderer, "Revisiting the Expected Cost of Solving uSVP and Applications to LWE," in *Advances in Cryptology – ASIACRYPT 2017* (T. Takagi and T. Peyrin, eds.), (Cham), pp. 297–322, Springer International Publishing, 2017.
45. M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *Journal of Mathematical Cryptology*, vol. 9, no. 3, pp. 169–203, 2015.

A Pump Simulator

To simulate the default G6K mode in the high dimension, we should use a Pump Simulator to simulate the change of the lattice basis. Our Pump simulator is designed under the assumption of HKZ, which means that the Gram-Schmidt lengths after Pump obeys HKZ assumption. We have compared the Gram-Schmidt lengths after Pump with the simulated one, and find out that the square error between them is close to 0. It shows that the Pump simulator is accurate while it takes dimension-for-free value as $f = \frac{d\ln(4/3)}{\ln(d/2\pi)}$ mentioned in section 2.4. Figure 7 shows the pump simulation result and the pseudo code is as Alg.5.

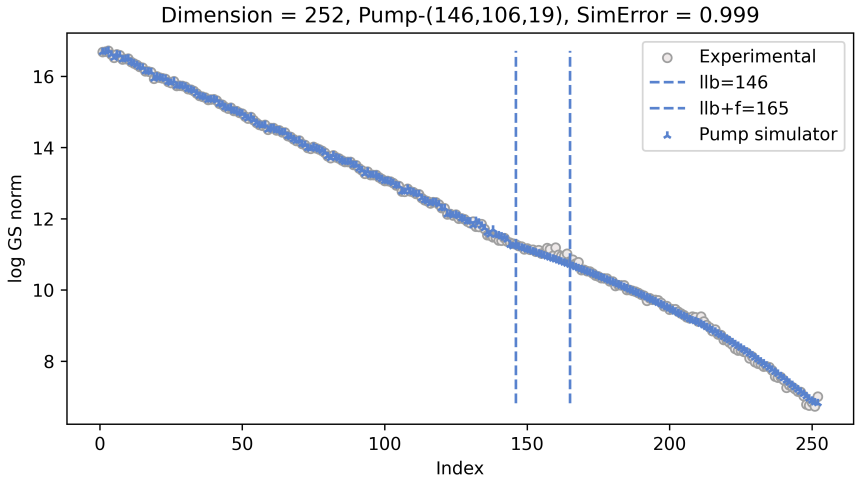


Fig. 7. Pump Simulation

B More experimental details about pnj-BKZ simulator

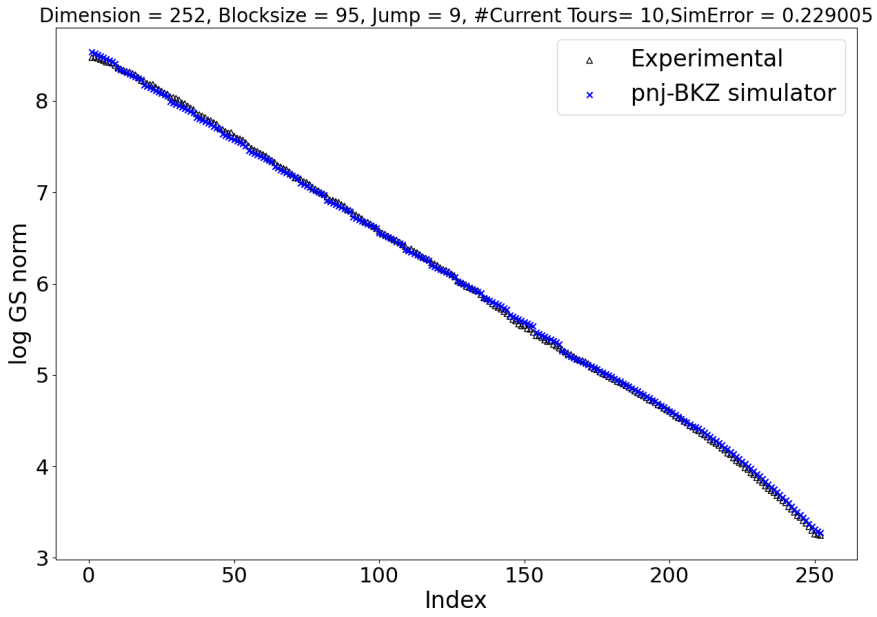
Here we give more verification experiments of our pnj-BKZ simulator, reducing ($n = 70, \alpha = 0.005$) LWE challenge lattice basis by pnj-BKZ with reduction parameter: ($\beta = 95, J = 9$) and ($\beta = 100, J = 12$) respectively, $\sharp$tours $\in [1, \cdots, 10]$. Here under the same reduction parameters, we do 20 times experiments. Figure 8 shows that our pnj-BKZ simulator fits well to the actual pnj-BKZ reduction result.

Figure 9 shows that when jump = 9 or jump = 12, even the tours increase to 10, the ratios $\frac{l_i''}{\mathrm{Sim}(l_i'')}$ are all between 0.975 and 1.05. It indicating that the pnj-BKZ simulator calculate the estimation value of the actual value $\|\mathbf{b}_i''^*\|$ by Eq. (1) can already reflect how the average of the norms of Gram-Schmidt vectors change during each tour's reduction of pnj-BKZ-(β, J). Therefore our pnj-BKZ simulator fits well with the actual pnj-BKZ reduction result.

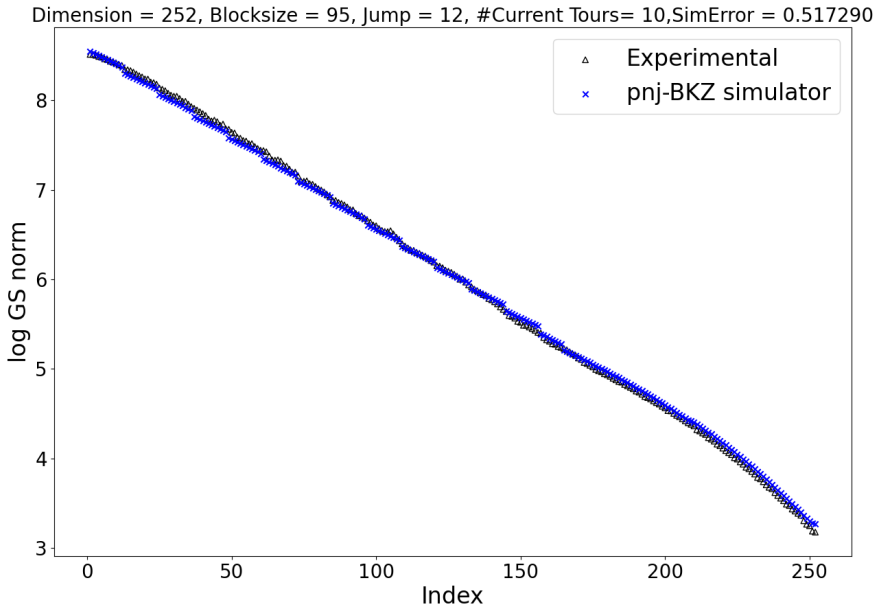
input : $(l_0, \cdots, l_{d-1})$,d, κ, β, f.
output: A prediction for the logarithms of the Gram-Schmidt norms $l'_i = \ln(\|\mathbf{b}'^*_i\|)$ after `Pump(`κ, β, f`)`.

$d_{\text{sieve}} \leftarrow \beta - f$;
for $\beta' \leftarrow d_{\text{sieve}}$ **to** d **do**
  if $\beta' < 40$ **then**
    $f' \leftarrow 0$;
  else
    $f' \leftarrow \frac{\beta' \ln 4/3}{\ln(\beta'/2\pi)}$;
  if $\beta' - f' \geq d_{\text{sieve}}$ **then**
    $\beta \leftarrow \beta'$; break;
for $i \leftarrow 0$ **to** 44 **do**
  $r_i \leftarrow$ average $\ln(\|\mathbf{b}^*_i\|)$ of a HKZ reduced random unit-volume 45-dimensional lattice;
for $i \leftarrow 45$ **to** β **do**
  $c_i \leftarrow \ln\left(\mathrm{V}_i(1)^{-1/i}\right) = \ln\left(\frac{\Gamma(i/2+1)^{1/i}}{\pi^{1/2}}\right)$;
for $k \leftarrow 0$ **to** $d - \beta - 1$ **do**
  $l'_k \leftarrow l_k$;
flag $\leftarrow$ **True**; //flag to store whether $L_{[k,d]}$ has changed
for $k \leftarrow d - \beta$ **to** $d - 46$ **do**
  $\beta' \leftarrow d - k$; $h \leftarrow d$; $\ln(\mathrm{V}) \leftarrow \sum_{i=1}^{h} l_i - \sum_{i=1}^{k-1} l'_i$;
  if *flag* = **True then**
    if $\ln(\mathrm{V})/\beta' + c_{\beta'} < l_k$ **then**
      $l'_k \leftarrow \ln(\mathrm{V})/\beta' + c_{\beta'}$; *flag* $\leftarrow$ *false*;
  else
    $l'_k \leftarrow \ln(\mathrm{V})/\beta' + c_{\beta'}$;
$\ln(\mathrm{V}) \leftarrow \sum_{i=1}^{h} l_i - \sum_{i=1}^{k-1} l'_i$;
for $k \leftarrow d - 45$ **to** $d - 1$ **do**
  $l'_k \leftarrow \frac{\ln(\mathrm{V})}{45} + r_{k+45-d}$;
for $k \leftarrow 0$ **to** $d - 1$ **do**
  $l_k \leftarrow l'_k$;
return $l_0, \cdots, l_{d-1}$;

Algorithm 5: Pump Simulator

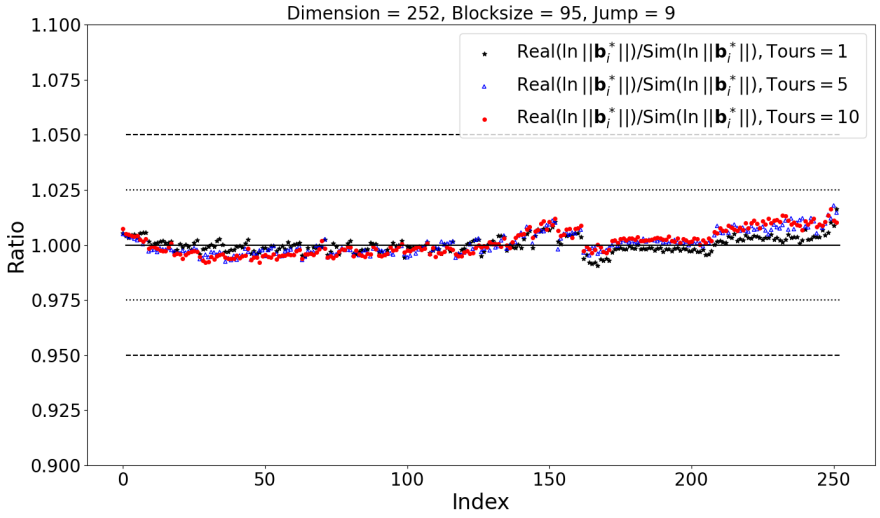


(a) $\beta = 95, jump = 9, \sharp\text{tours} = 10$

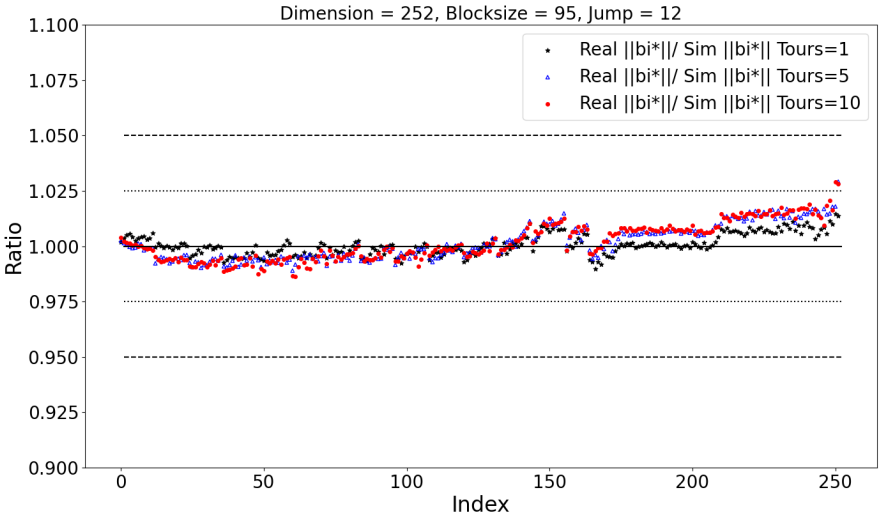


(b) $\beta = 95, jump = 12, \sharp\text{tours} = 10$

Fig. 8. Prediction effect of pnj-BKZ simulator. To verify the effectiveness of our pnj-BKZ simulator, we perform the following experiments, reducing $(n = 70, \alpha = 0.005)$ LWE challenge lattice basis by pnj-BKZ with reduction parameter: blocksize $\beta = 95$, jump size $J \in [1, \cdots, 16]$, $\sharp\text{tours} \in [1, \cdots, 10]$. Here under the same reduction parameters, we do 20 times experiments to obtain the average length of Gram-Schmidt vector.



(a) Jump=9



(b) Jump=12

Fig. 9. ratio $\frac{l_i''}{\mathrm{Sim}(l_i'')}$, $\beta = 95$. Run 10 tours ($\beta = 95$, $J = 9$ and $\beta = 95$, $J = 12$ respectively) of pnj-BKZ reduction on a lattice basis, and record the output of Gram-Schmidt vector lengths each tour. For each pnj-BKZ reduction parameter, we did experiments 20 times to obtain the average length of Gram-Schmidt vector.

C Practical time cost model of pnj-BKZ and Pump

To find the optimal progressive blocksize and jump size selection strategy for solving TU Darmstadt LWE challenges, it is necessary to construct pnj-BKZ and pump time cost models. However, the asymptotic complexity of the sieving does not match the actual cost well in the low-dimensional case[3] (dimension $\leq$ 128). The multithreading technology used in Pump will balance part of the time cost increases when the dimension of sieving increases. Therefore, we construct a practical time cost model by using the experimental method to test the running time of the Pump on a different lattice basis for finding the optimal reduction parameters of solving TU Darmstadt LWE challenges.

Here we need to point out that although the time-cost model based on the results of experiments can well fit the actual cost of running pnj-BKZ, using testing machines with different configurations will inevitably lead to changes in the time-cost model in low-dimensional cases. Therefore, we only use this experimentally constructed time-cost model when looking for the optimal progressive blocksize and jump size selection strategy for solving LWE challenges.

Besides, when we construct the actual time cost model by testing the time cost of pnj-BKZ on the specific machine, we find that each Pump in pnj-BKZ takes a different time cost as the figure 10(a) shown. Especially, the Pump cost increases under the incremental index smaller than $d-\beta+f$ and decreases after $d-\beta+f$ indices. It infers that for a fixed blocksize β, the average Pump cost in pnj-BKZ will increase with the growth of dimension d.

We can regard T_{pump} as a computational cost model of the $(\beta - f)$-dimensional progressive sieve, i.e.

$$\begin{aligned} T_{\text{pump}}(\beta) &= \sum_{j=\beta_0}^{\beta} T_{\text{sieve}}(j) = \sum_{j=\beta_0}^{\beta-f} 2^{c\cdot j+o(j)} = 2^{c\beta_0}\left(1+2^{c}+\cdots+2^{c(\beta-f-\beta_0)}\right) \\ &\leq 2^{c\beta_0}\cdot\frac{2^{c(\beta-f+1)+o(\beta-f+1)}}{1-2^{c}} = O\left(2^{c(\beta-f)}\right) \approx 2^{c(\beta-f)+c_1}, \end{aligned} \tag{4}$$

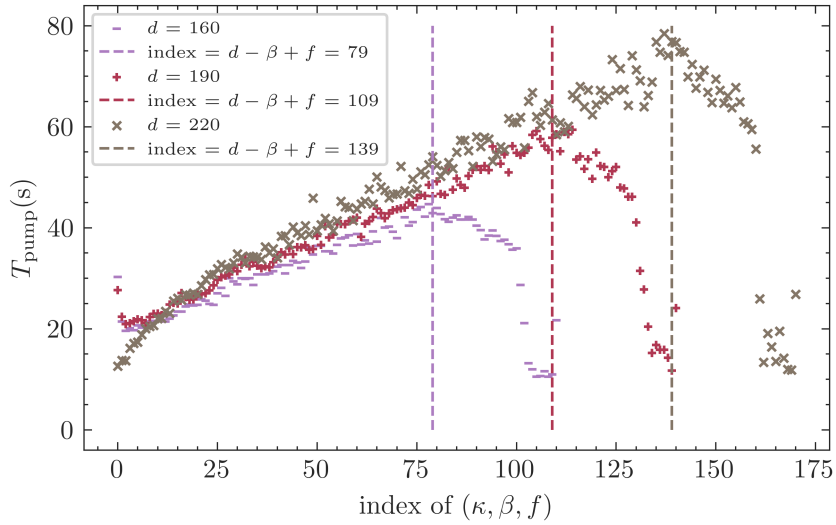
where β_0 is the dimension of initial sieving in Pump (In G6K β_0 is set to 30, and in G6K-GPU, it is set to 50), c and c_1 are the coefficients of the full sieve cost related to sieve dimension, $T_{\text{sieve}}(j)$ is the sieve cost with dimension j in dim-for-free.

However, we find that the asymptotic complexity of the sieving does not match the actual cost well in the low-dimensional case. While dimension is low, the number of threads used in Pump increases with the dimension, which balance out part of the time cost increase. So in low dimension, c might be much lower than the theoretical result.

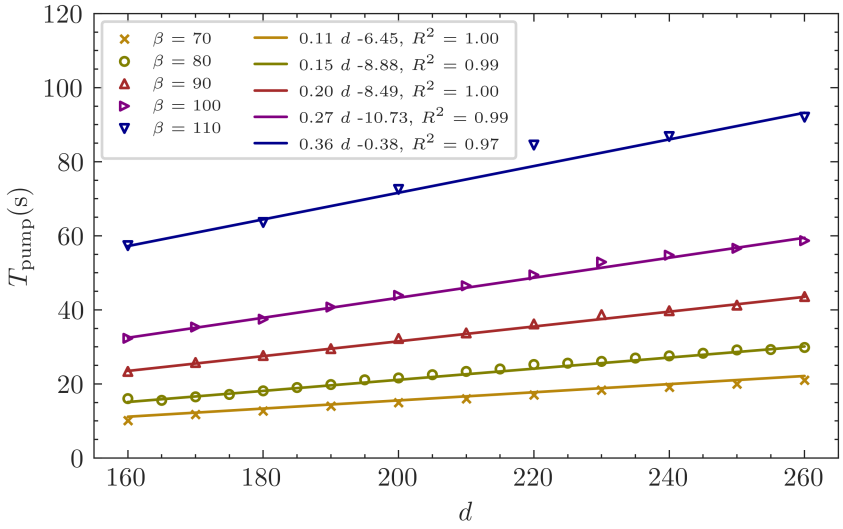
In order to accurately predict the unknown coefficients c and c_1 in the computational cost model, we use the experimental method to test the running time of Pump on different lattice basis corresponding to different TU Darmstadt LWE challenges and with different blocksizes β. The experimental results show that our computational cost model above can fit well with the actual cost of Pump.

Take $(\beta - f)$ as the independent variable, where f selected from equation (3) in section 2.4(Sieving Algorithms and Pump in G6K) of our main body. $\log_2 T$ is obtained from the experimental test as the dependent variable, and we use the least

[3] While dimension exceeds 128, the time cost for pump and pnj-bkz fits the theoretical value well, we can directly use the time cost model of trple_gpu sieve declared in [24].

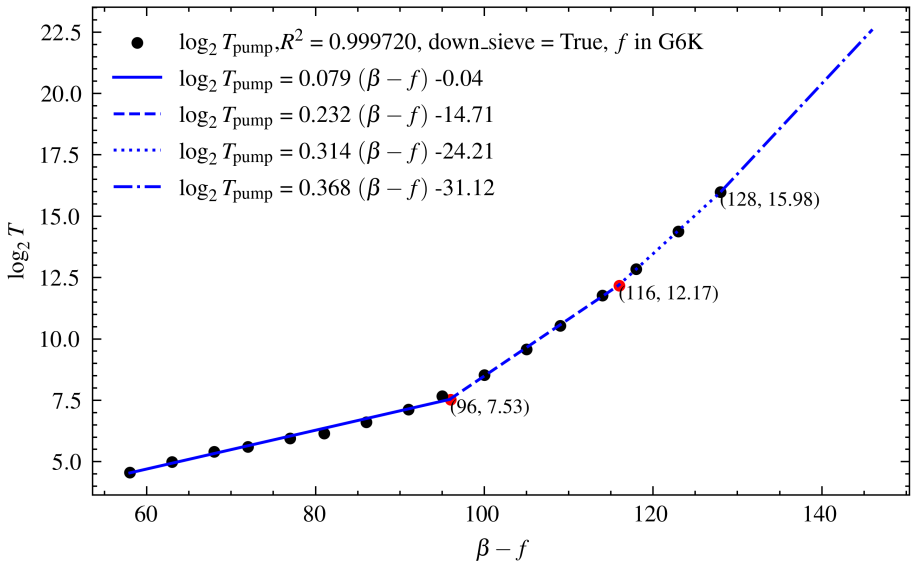


(a) Cost for each Pump under different index in pnj-BKZ-100 with one tour

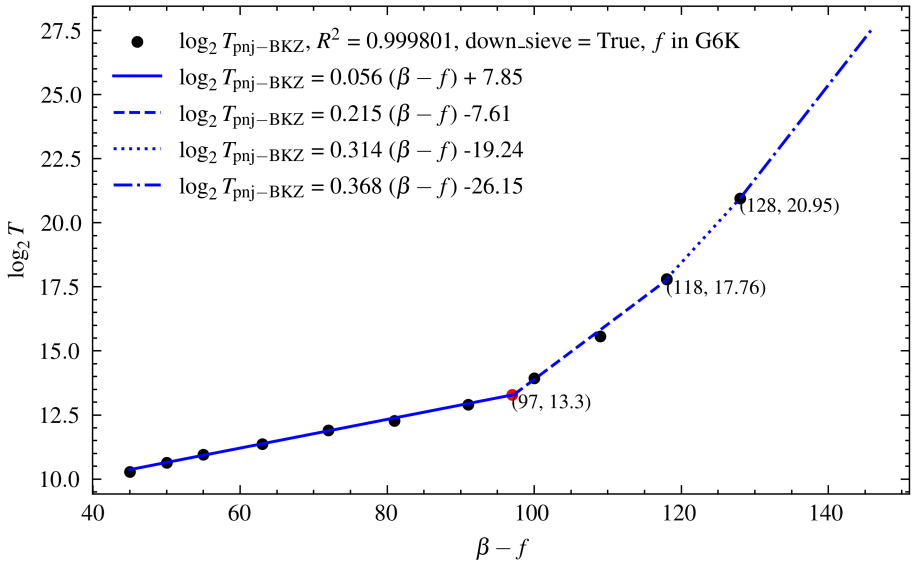


(b) Average Pump cost in pnj-BKZ-β grows linearly with d

Fig. 10. Pump cost under different indices in pnj-BKZ and average Pump costs in pnj-BKZ with change of d using Machine C.



(a) $\log_2 T_{\mathrm{pump}}$: Test Pump independently.



(b) $\log_2 T_{\mathrm{pnj-BKZ}}$: Average pnj-BKZ Cost. Test on random lattice basis from 180-dimension SVP Challengehttps://www.latticechallenge.org/svp-challenge.

Fig. 11. Pump and pnj-BKZ Cost Figure while $d = 180$, Pump Oracle = gpu_sieve, using Machine C: average T_{pump} in pnj-BKZ is lower than T_{pump} test directly, since in pnj-BKZ, the T_{pump} cost distributes uneven. The different functions from $\beta - f$ to $\log_2 T_{\mathrm{pump}}$ result from the saturation of threads in CPU/GPU.

squares fitting to find c and c_1. We use R^2 to denote the coefficient of determination (R squared) value above linear regression model. The coefficient of determination (R^2 or R squared) is a statistical measure in a regression model that determines the proportion of variance in the dependent variable that can be explained by the independent variable. Generally, the range of R^2 is $[0, 1]$ and when R^2 closer is to 1, the better the model fits the data.

From Figure 11(a), we can see that R^2 is close to 1. It means that the fitting effect is good. Figure 11(a) also shows that the logarithm of the computational cost of *Pump* is linearly correlated to $\beta - f$, where f is selected from dim4free function mentioned in equation (3) in section 2.4(Sieving Algorithms and Pump in G6K) of our main body.

Pnj-BKZ consists of a series of Pumps. If we regard pnj-BKZ as a combination of Pumps with equal cost, the computational cost of pnj-BKZ can be calculated by the sum cost of $\frac{d+2f-\beta}{J}$ progressive sieves on the $(\beta - f)$-dimension projection sublattice with $J = jump$. However, as the figure 10(a) shows, each Pump in pnj-BKZ takes a different cost. Especially, the Pump cost increases under the incremental index smaller than $d - \beta + f$ and decreases after $d - \beta + f$ indices. It infers that for a fixed blocksize β, the average Pump cost in pnj-BKZ will increase with the growth of dimension d. As Figure 10(b) shown, we have tested on 5 fixed blocksizes and proven that the average Pump cost in pnj-BKZ grows linearly with d.

We suggest that the average T_{pump} in pnj-BKZ is linear in d when d is small, and independent with d when d is large. Combining the functions among T_{pump}, blocksize β and dimension d, we can get the average Pump cost equation as

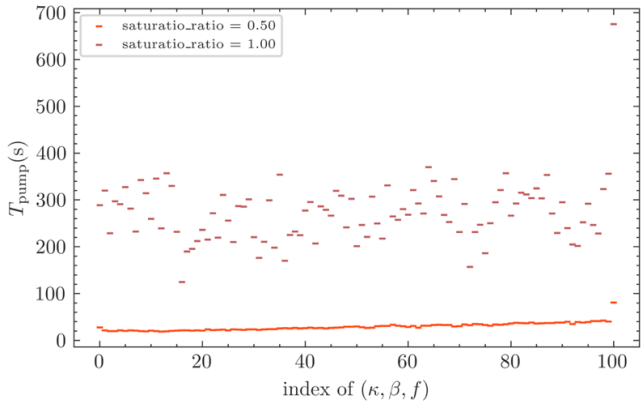
$$T_{\mathrm{pump}} = \min\left\{2^{c(\beta-f)+c_1} \cdot (c_2 \cdot d + c_3), 2^{c'(\beta-f)+c_1'}\right\} \tag{5}$$

where $2^{c\cdot(\beta-f)+c_1}$ is the Pump cost related to blocksize β and f satifsies the equation (3) in section 2.4(Sieving Algorithms and Pump in G6K) of our main body. For a lattice with dimension $d = 180$, the average Pump cost in pnj-BKZ can be simulated as Figure 11(b) and c_2 and c_3 can be computed by Figure 10(b) shown.

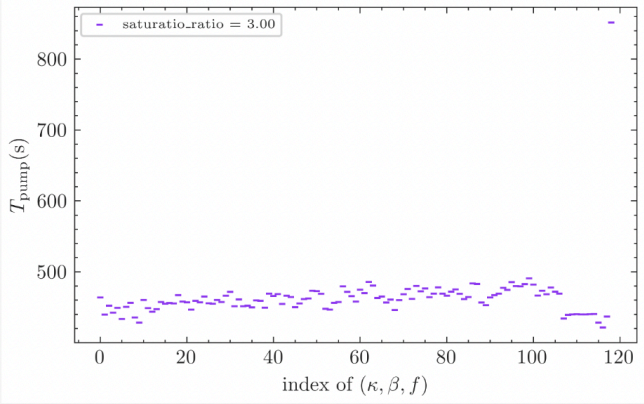
We believe that the relationship between average T_{pump} in pnj-BKZ and d is affected by the early termination condition in the implementation of sieving algorithms in Pump, where the algorithm stops when enough short vectors are generated. And we conducted the following experiments to verify our suspect.

We know that the key parameter controlling the early termination condition is the size of saturation ratio (saturation ratio is set to 0.5 by default in G6K). Therefore, we test the time cost of the pump on each projection sub-lattice under different saturation ratio. In order to remove the possible impact of the lift operation in the d4f technology, we set both d4f and f_{extra} to 0 in Figure 12(a). From Figure a we can see that the phenomenon that the pump time cost increases with the increase of the initial index of the projected sub-lattice is no longer obvious when saturation ratio is set to 1. In other words, when the early termination condition is removed, the time cost of the pump increasing phenomenon will disappear. In fact even if the d4f technique is used, as long as the early termination condition is removed, the the time cost of the pump increasing phenomenon will disappear. See Figures 12(b) and 12(c) for details.

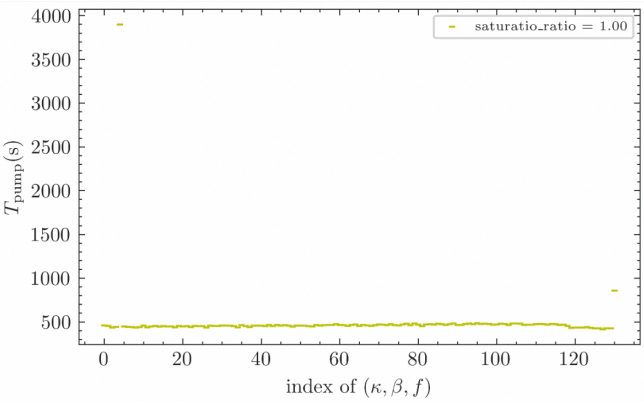
If d is large enough, we suppose that the algorithm stops only when all vectors are reduced, then the time cost of the Pump will achieve the theoretical complexity, so the average Pump cost with the growth of d will also achieve an upper bound $T_{\mathrm{pnj-BKZ}}^{(max)} = 2^{c'(\beta-f)+c_1'}$. However, we are currently not able to calculate the exact value of c' and c_1'. Besides, We believe that the reason for the gradual decrease in



(a) $d = 180$, $\beta = 80$, d4f $= 0$, $f_{\text{extra}} = 0$.



(b) saturation ratio=3.0, $d = 180$, $\beta = 100$, d4f $= 19$, $f_{\text{extra}} = 0$.



(c) saturation ratio=1.0 $d = 180$, $\beta = 100$, d4f $= 19$, $f_{\text{extra}} = 12$.

Fig. 12. The influence of early termination condition.

pump cost corresponding to the last $d - \beta + f$ is the gradual decreasing dimension sieving in these pumps.

While we construct the practical time cost model of `Pump` to find the optimal reduction parameter, we found that the time cost of each `Pump` increases linearly with the growth of the index (index $< d - f$) in each tour of PnjBKZ. Currently, we only know that this phenomenon is caused by the early termination condition, but we don't know how the early termination condition affects the time cost of the `Pump`, and we plan to find out the reason in the future.

D Choosing the number of LWE Samples

BKZ-only mode is the mainstream method for estimating the security of an LWE-based crytosystem at current. It uses Kannan's Embedding technique to reduce the LWE problem to the uSVP_γ problem and uses the GSA assumption to simulate the change after a BKZ-β reduction. Its evaluation method was firstly proposed by Erdem Alkim *et al.*. in [13] and has been proved the correctness in [44], which has both given a lower bound of LWE samples and a blocksize β. We rename it "2016 Estimation from GSA for LWE" (refer to as 2016 Estimate).

In order to solve the LWE problem, the first thing we need to do is to determine the number of LWE instances to construct the lattice basis described in the primal attack. The strategy to select the number of LWE instances in 2016 Estimate is to find the number of LWE instances m so that the following inequality holds and the value of β is minimal. Let $d = m + 1$, n be the dimension of LWE instance, then

$$\min_{\beta \in \mathbb{N}} \left\{ T_{\text{BKZ}}(\beta) : \sigma\sqrt{\beta} \leq \delta\left(\beta\right)^{2\beta - d - 1} \cdot q^{\frac{d-n-1}{d}} \right\}. \tag{6}$$

The strategy in 2016 Estimate is to find m so that the LWE problem can be solved with the least time cost when using a fixed blocksize of BKZ-β algorithm to solve it.

In G6K, its estimation method simulates a two-stage strategy. Their main difference from ours is that its two-stage strategy contains two tours of pnj-BKZ with a fixed blocksize β simulated from GSA assumption and a progressive sieve algorithm in dimension d_{svp}. It simulates the above scenario and try to find the minimal cost of (β, d_{svp}) from

$$\min_{\beta, d_{\text{svp}} \in \mathbb{N}} \left\{ 2 \cdot T_{\text{BKZ}}(\beta) + \text{PSC}(d_{\text{svp}}) : \|\pi_{d-d_{\text{svp}}}(\mathbf{v})\| \leq \text{GH}(\mathcal{L}_{\pi[d-d_{\text{svp}}]}) \right\}, \tag{7}$$

where $c = 0.349$ in G6K-CPU and $c = 0.292$ in G6K-GPU.

Our strategy for solving the LWE problem is also simulating a two-stage strategy. In the first stage, it will call the pnj-BKZ simulator to simulate the basis after a series of pnj-BKZ. In the second stage, it tries to find the approximate shortest vector by Pump. Based on the estimation scheme in the default G6K described above, we modify the time cost of two pnj-BKZs and a progressive sieve to the time cost of serial pnj-BKZs following the blocksize strategy and a progressive sieve. Besides, we use the new Pump estimation scheme (as described in Algorithm: Pump estimation in LWE) to simulate the norm of the target vector. Let $P(d_{\text{svp}}) = \Pr\left[y \leftarrow \sigma^2 \chi^2_{d_{\text{svp}}} \middle| y \leq \left(\text{GH}\left(\mathcal{L}_{\pi[d-d_{\text{svp}}:d]} \right) \right)^2 \right]$. Thus, the inequality becomes

$$\min_{\beta, d_{\text{svp}} \in \mathbb{N}} \left\{ T_{\text{pnjBKZs}}\left(\mathbf{B}\right) + \text{PSC}\left(d_{\text{svp}}\right) : P(d_{\text{svp}}) \geq P_{\text{success}} \right\}, \tag{8}$$

where δ is the basis quality after pnj-BKZs. $T_{\text{pnjBKZs}}(\mathbf{B})$ will respectively call BSSA and EnumBS to calculate the corresponding computational cost. To minimize the number of attempts, we narrow the range of m to $[m_0 - \tau, m_0 + \tau]$, where m_0 is the number of samples chosen in the estimation of default G6K and set a maximum search field range τ. We use a dichotomization to find an m with minimal β and d_{svp} satisfying the inequality (8). Furthermore, the concrete process is as the Algorithm 6.

input: n, q, α, m_{all}, β_{bound}, $d^{(\text{svp})}_{\text{bound}}$, τ, $\mathbf{A}^{m_{all}\times n}$, $\mathbf{b}^{m_{all}\times 1}$;
output: $S_{\min}$, $T_{\min}$, m;
$\sigma, T_{\min}, \mathsf{mRange} \leftarrow \alpha q, +\infty, \{\}$;
$m_0 \leftarrow$ LWE samples estimation in G6K as formula (7);
$m_{\max}, m_{\min} \leftarrow \max\{m \textbf{ satisfies } \text{equation (7)}\}, \min\{m \textbf{ satisfies } \text{equation (7)}\}$;
while $\tau \neq 0$ **do**
  Construct $\mathbf{B}$ by $\left(\mathbf{A}^{m_0\times n}, \mathbf{b}^{m_0\times 1}, q\right)$;
  $S_{\min}, T_{\min} \leftarrow \mathtt{EnumBS(rr(}\mathbf{B}\mathtt{)}, m_0+1, \sigma^2 m_0+1, J\mathtt{)}$;
  $m_1 \leftarrow m_0$;
  for $m \in \{\max\{n, m_0-\tau\}, \min\{m_{all}, m_0+\tau\}\}$ **do**
    if $m \geq m_{\min}$ **and** $m \leq m_{\max}$ **then**
      $d \leftarrow m+1$, $M \leftarrow \sigma^2 m + 1$;
      Construct $\mathbf{B}$ by $\left(\mathbf{A}^{m\times n}, \mathbf{b}^{m\times 1}, q\right)$;
      $S, T_{\text{total}} \leftarrow \mathtt{EnumBS(rr(}\mathbf{B}\mathtt{)}, d, M, J\mathtt{)}$;
      if $T_{\min} < T_{\text{total}}$ **then**
        $S_{\min}, T_{\min}, m_1 \leftarrow S, T_{\text{total}}, m$;
  if $m_1 = m_0$ **then**
    $\tau \leftarrow \lfloor \frac{\tau}{2} \rfloor$;
  $m_0 \leftarrow m_1$;
return $S_{\min}, T_{\min}, m_0$;

Algorithm 6: Our LWE Samples Selection Algorithm

Using the optimization strategy for LWE instance number selection, we can solve challenges faster than G6K default strategy. See the table 8.

Table 8. LWE samples improvement simulated result with $jump = 1$.

(n,α)	G6K's m	Our m	$\text{Cost}_{\text{new}}/\text{Cost}_{\text{old}}$
(50,0.025)	218	216	99.98%
(55,0.020)	229	234	98.76%
(60,0.015)	240	246	99.30%
(90,0.005)	305	312	95.11%

E A refined Two-step security estimator for solving LWE

In this section, we will give the details of our Two-step security estimator for solving LWE. Firstly, we construct our Two-step LWE concrete hardness estimator in Section E.1. Then, we give the experiments to verify the accuracy of our Two-step LWE estimator in Section E.2. Finally, we compare the Two-Step mode estimator using different reduction strategies (and considering whether to use the technology of list-decoding or not) with the leaky-lwe estimator in Section E.3.

In addition, we will analyze the influence brought by optimized blocksize and jump selection on the hardness of LWE in Section **??**.

E.1 Construction of the Two-step LWE concrete hardness estimator

The Two-step mode was first proposed by [12], which calls a series of BKZ first for lattice reduction and calls an enumeration algorithm to find the target vector at last. In this paper, since NIST selected PQC schemes [1, 2] uses leaky-LWE-Estimator to evaluate the security strength of their schemes, and we want to know the impact of our Two-step mode of solving LWE with optimized blocksize and jump selection strategy on the security strength of the NIST selected PQC schemes, we specialise to construct Two-step LWE concrete hardness estimator mainly based on the leaky-LWE-Estimator. In fact, based on other security evaluators such as Martin's evaluator [45] to construct our Two-step LWE concrete hardness estimator can also obtain similar conclusions that our Two-step mode of solving LWE with optimized blocksize and jump selection strategy will result in a decrease of the estimated security bits.

Specifically, we use a Two-step mode adapted to pnj-BKZ and Pump. It calls a series of pnj-BKZ to reduce the basis first and finds a good timing to use a Pump algorithm to search the approximate shortest vector in the end, to solve the LWE problem. The concrete process is as Algorithm 1. By our pnj-BKZ simulator Algorithm 2 and Pump sieving dimension estimation Algorithm 3, we can guarantee that the last Pump outputs the required target vector. More detail about verification experiments can be seen in Section E.2.

Leaky-LWE-Estimator Therefore let's briefly review the leaky-LWE-Estimator. The reduction strategy S used in Algorithm indicates the blocksize and jump selection strategy during the reduction step. There is a trivial reduction strategy $\mathsf{S}_0 = \{(\beta_i = i+1,\ J_i = 1) \mid i = 1, ..., k\}$, here k can be set as the integer which smaller than the dimension of embedding lattice d. In fact, the LWE solving strategy considered by leaky-LWE-Estimator [29] is the same as S_0. The difference is that leaky-LWE-Estimator [29] will use the solving success probability to determine the final k value. A natural thought is whether there is a better blocksize and jump selection strategy to make the total cost of solving LWE more smaller, which also is an open problem in Section 5.3 of [2]. In order to evaluate the impact of optimized blocksize and jump selection strategy and Two-step mode on the security strength of NIST candidate schemes, we give a new refined security hardness estimator for solving LWE through the primal attack in the following Section **??**.

Before we introduce the influence of optimized blocksize and jump selection strategy, let us first illustrate that using the Two-step solving strategy whose reduction step uses the trivial reduction strategy S_0 can already decrease the security bits of these NIST selected PQC schemes [1, 2].

Unlike classical LWE hardness estimation [13], leaky-LWE-Estimator considers the sum square of the length of the target vector as a random variable that follows the chi-square distribution. We think it is more accurate to describe the true situation of primal attack since $(\mathbf{s}, \mathbf{e})$ corresponding to each LWE instance is randomly selected from the Gaussian distribution. So we also use the same idea that we treat the length of the target vector as a random variable rather than some fixed expected value.

Besides, Leaky-LWE-Estimator considers the progressive BKZ-only mode. Let W be the event of solving LWE during the run of Progressive BKZ, W_β the probability of being able to solve LWE during the round with blocksize β and $\mathrm{F}_\beta = \neg \mathrm{W}_\beta$. They implicitly partition W as follows:

$$\Pr[\mathrm{W}] = \Pr[\mathrm{W}_3] + \Pr[\mathrm{W}_4 \wedge \mathrm{F}_3] + \Pr[\mathrm{W}_5 \wedge \mathrm{F}_4 \wedge \mathrm{F}_3] + \cdots = \sum_{\beta=3}^{d} \Pr\left[\mathrm{W}_\beta \wedge \bigwedge_{j=3}^{\beta-1} \mathrm{F}_j\right]$$

Here $\Pr[\mathrm{W}_\beta] = \Pr\left[x \leftarrow \chi_\beta \middle| x \leq \mathsf{rr}_{[d-\beta:d]}\right]$, and $\mathsf{rr}_{[d-\beta:d]}$ is the length of the first Gram-Schmidt vector of projective sub-lattice $L_{[d-\beta:d]}$ of current lattice basis which has been reduced by Progressive BKZ with reduction strategy $\mathsf{S}_0 = \{(\beta_i = i + 2, J_i = 1) \mid i = 1, ..., \beta\}$.

Then leaky-lwe estimator based on a Heuristic assumption that events W_i and F_j for $i \neq j$ are independent. Therefore, they calculate $P[\mathrm{W}]$ by Equation 9. See the discussion in Section 4.1 of [33] or the implement of leaky-lwe estimator: Algorithm 7 for more details.

input: d;
output: $\bar{\beta}$;
$p_{tot} \leftarrow 0,\ \bar{\beta} \leftarrow 0$
profile $\leftarrow$ GSA profile of an LLL reduced, rank d, LWE instance basis
for $\beta \leftarrow 3$ **to** d **do**
  profile $\leftarrow$ BKZ 2.0 simulator(**profile**, β);
  $p_{lift} \leftarrow \mathrm{P}[t$ recovered in $\lfloor d/\beta \rfloor$ rounds $\mid \pi_{d-\beta+1}(t)$ recovered this round$]$
  $p_{rec} \leftarrow \mathrm{P}[x \leftarrow \chi^2_\beta : x \leftarrow \chi^2_\beta : x \leq$ **profile**$[d - \beta + 1]]$
  $p_{new} \leftarrow (1 - p_{tot}) \cdot p_{rec} \cdot p_{lift}$
  $\bar{\beta} \leftarrow \bar{\beta} + p_{new} \cdot \beta$
  if $p > 0.999$ **then**
    break
return $\bar{\beta}$

Algorithm 7: Concrete hardness estimator of LWE in [29]

Here, the χ^2_β is the chi-squared distribution with β degrees of freedom is the distribution of a sum of the squares of β independent standard normal random variables.

$$\Pr[\mathrm{W}] = \sum_{\beta=3}^{d} \Pr\left[\mathrm{W}_\beta \wedge \bigwedge_{j=3}^{\beta-1} \mathrm{F}_j\right] = \sum_{\beta=3}^{d} \Pr\left[\mathrm{W}_\beta\right] \cdot \Pr\left[\bigwedge_{j=3}^{\beta-1} \mathrm{F}_j\right] \tag{9}$$

Equation 9 can be established only when W_i and event F_j for $i \neq j$ are independent events, otherwise $\Pr\left[\mathrm{W}_\beta \wedge \bigwedge_{j=3}^{\beta-1} \mathrm{F}_j\right] \neq \Pr[\mathrm{W}_\beta] \cdot \Pr\left[\bigwedge_{j=3}^{\beta-1} \mathrm{F}_j\right]$. From the line 4 of Algorithm 7 we know that the quality of lattice basis will be improved by every time a new bigger lattice reduction. The Heuristic assumption that events W_i and F_j for $i \neq j$ are independent that leaky-lwe estimator based on, is reasonable to some extent, if we assume that the lattice basis will be re-randomized each time it is reduced by a stronger BKZ reduction. Below we reformulate this assumption formally:

Heuristic 6 *The lattice basis is randomized by each time of a new bigger BKZ-β reduced. Then every events W_i and F_j for $i \neq j$ are independent.*

More specifically, if we set event E_β for $\beta \in \{3, ...d\}$ as the event that solving LWE during the process of running Progressive BKZ: BKZ-3,..., BKZ-β. Then $\bigcup_{i=3}^{\beta}\left((\mathrm{W}_i) \wedge \bigwedge_{j=3}^{i-1} \mathrm{F}_j\right) = \mathrm{E}_\beta$ and $\bigwedge_{j=3}^{\beta-1} \mathrm{F}_j = \neg \mathrm{E}_{\beta-1}$. Then we obtain the same probability accumulation calculation recursive Equation 10 as that in the line 7 of of Algorithm 7.

$$\Pr[\mathrm{E}_\beta] = \Pr[\mathrm{E}_{\beta-1}] + \Pr[\mathrm{W}_\beta] \cdot (1 - \Pr[\mathrm{E}_{\beta-1}]) \tag{10}$$

Equation 10 holds if and only if Heuristic 6 is established. Since only when events W_i and event F_j for $i \neq j$ are completely independent events, one can write $\Pr\left[\bigwedge_{j=3}^{\beta-1} \mathrm{F}_j\right] = 1 - \Pr[\mathrm{E}_{\beta-1}]$.

Then based on Equation 10 leaky-LWE-Estimator calculates the accumulation probability $\Pr[\mathrm{E}_\beta]$ for $\beta \in \{3, ...\}$ of solving LWE when using progressive BKZ with reduction strategy $\mathsf{S}_0 = \{(\beta_i = i + 2, J_i = 1) \mid i = 1, ..., \beta\}$. And leaky-LWE-Estimator will output $\bar{\beta}$ when a certain β s.t $P[\mathrm{E}_\beta] > 0.999$. Here $\bar{\beta} = \sum_{i=3}^{\beta} i \cdot P[\mathrm{E}_i]$.

Finally leaky-LWE-Estimator will call the Gate-count algorithm in [30] to calculate the number of gates (time cost): $gates(\bar{\beta}) = C^2 \cdot \mathrm{agps20gates}(\bar{\beta} - d_{4f}(\bar{\beta}))$ and memory cost: $bits_{pump}(\bar{\beta}) = 8(\bar{\beta} - d_{4f}(\bar{\beta})) \cdot \mathrm{agps20vectors}(\bar{\beta} - d_{4f}(\bar{\beta}))$ for solving the LWE respectively. Here the Gate-court algorithm [30] can analyze the cost of sieving with a classical and quantum circuit and $C = \frac{1}{1-2^{-0.292}}$, a constant used to simulate the time cost of progressive sieving when BDGL16 sieving [25] is used and progressive BKZ blocksize. More detail about functions agps20gates() can be seen in [43] and $d_{4f}()$ is Equation 2.4.

Two-step-LWE-estimator We use similar notations: W be the event of solving LWE during running Progressive pnj-BKZ or the final high-dimension Pump of Two-step mode, $\mathrm{W}^{(1)}_{(\beta,J)}$ be the event of solving LWE by using pnj-BKZ-(β, J), $F^{(1)}_{(\beta,J)} = \neg \mathrm{W}^{(1)}_{\beta}$ and $\mathrm{W}^{(2)}_{(d_{svp})}$ as the event that a d_{svp}-dimension Pump solved LWE. Here $\Pr[\mathrm{W}^{(1)}_{(\beta,J)}] = \Pr\left[x \leftarrow \chi_\beta \middle| x \leq \mathsf{rr}_{[d-\beta:d]}\right]$, and $\mathsf{rr}_{[d-\beta:d]}$ is the length of the first Gram-Schmidt vector of projective sub-lattice $L_{[d-\beta:d]}$ of current lattice basis which has been reduced by Progressive BKZ with reduction strategy $\mathsf{S} = \{(\beta_i,\ J_i) \mid i = 1, ..., \beta\}$. In Two-step mode we partition W as:

$$\Pr[\mathrm{W}] = \Pr[\mathrm{W}^{(1)}_{(\beta_1,J_1)}] + \Pr[\mathrm{W}^{(1)}_{(\beta_2,J_2)} \wedge \mathrm{F}^{(1)}_{(\beta_1,J_1)}] + \Pr[\mathrm{W}^{(1)}_{(\beta_3,J_3)} \wedge \mathrm{F}^{(1)}_{(\beta_2,J_2)} \wedge \mathrm{F}^{(1)}_{(\beta_1,J_1)}] + \ldots$$
$$+ \Pr\left[\mathrm{W}^{(1)}_{(\beta_{end},J_{end})} \wedge \bigwedge_{j=1}^{end-1} \mathrm{F}^{(1)}_{(\beta_j,J_j)}\right] + \Pr\left[\mathrm{W}^{(2)}_{(d_{svp})} \wedge \bigwedge_{j=1}^{end} \mathrm{F}^{(1)}_{(\beta_j,J_j)}\right] =$$
$$\sum_{i=1}^{end} \Pr\left[\mathrm{W}^{(1)}_{(\beta_i,J_i)} \wedge \bigwedge_{i>1,j=1}^{i-1} \mathrm{F}^{(1)}_{(\beta_j,J_j)}\right] + \Pr\left[\mathrm{W}^{(2)}_{(d_{svp})}\right] \cdot \Pr\left[\bigwedge_{j=1}^{end} \mathrm{F}^{(1)}_{(\beta_j,J_j)}\right] \tag{11}$$

Here $\mathrm{W}^{(2)}_{(d_{svp})}$ means during the process of Pump, d_{svp}-dimension progressive sieving finds the projection vector of the target vector and $\mathrm{F}^{(2)}_{d_{svp}} = \neg\mathrm{W}^{(2)}_{(d_{svp})}$. Event $\mathrm{W}^{(2)}_{(d_{svp})}$ happened means all pnj-BKZ(β, J) in reduction step fail to find the target vector, other else it will not call the final high-dimension Pump. So Event $\mathrm{W}^{(2)}_{(d_{svp})}$ is independent with all events $\mathrm{F}^{(1)}_{(\beta_j,J_j)}$. The value of d_{svp} will be set to solve this LWE with a probability above 0.999. Set *end* as the index of the last block in the pnj-BKZ reduced sequence and d_{start} is the dimension of the initial projection sub-lattice sieved by Pump.

Besides, set event $\mathrm{E}^{(1)}_i$ for $i \in \{1, 2, ...\}$ as the event that solving LWE during the process of running Progressive pnj-BKZ: pnj-BKZ(β_1, J_1),..., pnj-BKZ(β_i, J_i). based on Heuristic 6, we have $\Pr\left[\mathrm{W}^{(1)}_{(\beta_i,J_i)} \wedge \bigwedge_{i>1,j=1}^{i-1} \mathrm{F}^{(1)}_{(\beta_j,J_j)}\right] = \Pr\left[\mathrm{W}^{(1)}_{(\beta_i,J_i)}\right] \cdot \Pr\left[\bigwedge_{i>1,j=1}^{i-1} \mathrm{F}^{(1)}_{(\beta_j,J_j)}\right]$ and Equation 12.

$$\Pr[\mathrm{E}^{(1)}_i] = \Pr[\mathrm{E}^{(1)}_{i-1}] + \Pr[\mathrm{W}^{(1)}_{(\beta,J)}] \cdot \left(1 - \Pr[\mathrm{E}^{(1)}_{i-1}]\right) \tag{12}$$

We will use Equation 12 to calculate the cumulative probability of solving LWE during reduction step, see line 8 of Algorithm 10.

However, how to calculate the probability of solving LWE when running the final Pump is one of the main differences between our Two-step LWE estimator and leaky-lwe estimator. First of all, the leaky-lwe estimator does not consider solving uSVP by using Pump. Secondly, in leaky-lwe estimator, they calculate the cumulative solving probability by Equation (10) which holds only when events W_i and F_j for $i \neq j$ are independent. According to the Heuristic 6, we suppose that only when the quality of lattice basis has been improved by a stronger reduction, the lattice basis will be re-randomized so that events $\mathrm{W}^{(2)}_{(i)}$ and $\mathrm{F}^{(2)}_{(j)}$ for $i \neq j$ can be supposed as independent. In fact during running the final Pump to solve LWE, although the Pump is a progressive sieve, the lattice basis will not change during sieving. Therefore, the similar Heuristic assumption that events $\mathrm{W}^{(2)}_{(i)}$ and $\mathrm{F}^{(2)}_{(j)}$ for $i \neq j$ are independent may not established. Since unlike progressive BKZ every time a new bigger BKZ reduction will re-randomize the lattice basis, the lattice basis does not change every time the sieving area $L_{[i:d]}$ increases 1 dimension by extend left operation.

On the contrary, instead of considering that events $\mathrm{W}^{(2)}_{(i)}$ and $\mathrm{F}^{(2)}_{(j)}$ for $i \neq j$ are independent, we consider that there is an inclusive relationship between $\mathrm{W}^{(2)}_{(i)}$ and $\mathrm{W}^{(2)}_{(j)}$ for $j \leq i$, i.e $\mathrm{W}^{(2)}_{(i)} \supseteq \mathrm{W}^{(2)}_{(j)}$. Since the lattice basis will not change during the progressive sieving of Pump and running an i-dimension progressive sieving, it will definitely running a j-dimension progressive sieving at first, for $j \leq i$.

Setting event $\mathrm{E}^{(2)}_{(\beta)}$ as until the sieving dimension of a Pump increase to β, this Pump find the projection of target vector which means during β-dimension progressive

sieving all the sieving dimension smaller than β failed to find the target vector. We give following Heuristic assumption.

Heuristic 7 *For* $i \in \{2, ..., d_{svp}\}$, $\mathrm{W}^{(2)}_{(i)} \supseteq \mathrm{W}^{(2)}_{(i-1)} \supseteq \mathrm{W}^{(2)}_{(i-2)} \cdots \supseteq \mathrm{W}^{(2)}_{(2)}$. *Then* $\mathrm{E}^{(2)}_{(i)} = \mathrm{W}^{(2)}_{(i)} - \mathrm{W}^{(2)}_{(i-1)}$.

According to Heuristic 7 we can calculate the cumulative probability of solving LWE by using a high-dimension Pump by Equation 13, see the line 12 of of Algorithm 10 for more details.

$$\Pr\left[\mathrm{E}^{(2)}_{(d_{svp})}\right] = \Pr\left[\mathrm{W}^{(2)}_{(d_{svp})}\right] - \Pr\left[\mathrm{W}^{(2)}_{(d_{svp}-1)}\right] \tag{13}$$

The difference between our Two-step-LWE-estimator and leaky-LWE-Estimator In addition to consideration of Pump, another difference between our Two-step-LWE-estimator and Leaky-LWE-Estimator is the way of calculating concrete hardness bits of LWE.

Leaky-LWE-Estimator use Algorithm 7 to obtain the expected value of BKZ blocksize $\bar{\beta}$ for solving LWE to estimate the hardness of LWE. We think it is inaccurate. In the description of [33], we can get that

$$\bar{\beta} = \sum_{\beta=3}^{d} \beta \cdot \Pr\left[\mathrm{W}_\beta \wedge \bigwedge_{j=3}^{\beta-1} \mathrm{F}_j\right],$$

then the gate count of $\bar{\beta}$ is

$$\begin{aligned}
\mathrm{gate}(\bar{\beta}) &= C^2 \cdot \mathrm{agps20gates}(\bar{\beta} - d_{4f}(\bar{\beta})) \\
&= C^2 \cdot \mathrm{agps20gates}(\sum_{\beta=3}^{d} \beta \cdot \Pr\left[\mathrm{W}_\beta \wedge \bigwedge_{j=3}^{\beta-1} \mathrm{F}_j\right] - d_{4f}(\sum_{\beta=3}^{d} \beta \cdot \Pr\left[\mathrm{W}_\beta \wedge \bigwedge_{j=3}^{\beta-1} \mathrm{F}_j\right])) \\
&= C^2 \cdot \mathrm{agps20gates}(\sum_{\beta=3}^{d} (\beta - d_{4f}(\beta)) \cdot \Pr\left[\mathrm{W}_\beta \wedge \bigwedge_{j=3}^{\beta-1} \mathrm{F}_j\right])) \\
&= C^2 \prod_{\beta=3}^{d} \mathrm{agps20gates}(\beta - d_{4f}(\beta))^{\Pr\left[\mathrm{W}_\beta \wedge \bigwedge_{j=3}^{\beta-1} \mathrm{F}_j\right])} \\
&= \prod_{\beta=3}^{d} G_{\mathrm{proBKZ}}(\beta - d_{4f}(\beta))^{\Pr\left[\mathrm{W}_\beta \wedge \bigwedge_{j=3}^{\beta-1} \mathrm{F}_j\right])}.
\end{aligned}$$

$\mathrm{gate}(\bar{\beta})$ in [33] is not an average cost for progressieve BKZ. Thus, instead of calculating the expected value of BKZ blocksize $\bar{\beta}$ then calculate the number of gates (time cost) by $\bar{\beta}$: $\mathrm{gate}(\bar{\beta}) = C^2 \cdot \mathrm{agps20gates}(\bar{\beta} - d_{4f}(\bar{\beta}))$ and memory cost: $\mathrm{bits}_{\mathrm{pump}}(\bar{\beta}) = 8(\bar{\beta} - d_{4f}(\bar{\beta})) \cdot \mathrm{agps20vectors}(\bar{\beta} - d_{4f}(\bar{\beta}))$ for solving the LWE

However, we choose directly calculate the expected cost to give a more accurate security estimation as it shown in Algorithm 8.

More specifically, we consider the total gates count for Two-step mode of solving LWE divided by two parts: G_1 and G_2. Here, G_1 is the number of gates of the case that if we solving the u-SVP during running the reduction step without entering the search step and G_2 is the number of gates of the case that if we solving the u-SVP

input : n, m, q, χ, P_{success};
output: GB;

1 $\mathsf{GB} \leftarrow (0, 0)$; $p_{\text{tot}} \leftarrow 0$;
2 $\mathsf{rr} \leftarrow$ GSA profile of an LLL reduced $\text{LWE}_{n,m,q,\chi}$ instance;
3 **for** β **from** β_0 **to** d **do**
4 $\quad \mathsf{rr} \leftarrow \mathsf{SimBKZ}(\mathsf{rr}, \beta, 1)$;
5 $\quad P_{\text{suc}}(\beta) \leftarrow \Pr\left[x \leftarrow \chi_\beta \middle| x \leq \mathsf{rr}_{[d-d_{\text{svp}}:d]}\right]$;
6 $\quad \mathsf{GB}_{\text{cum}} \leftarrow (\sum_{b=\beta_0}^{\beta} \mathsf{G}_{\text{pnjBKZ}}(b), \mathsf{B}_{\text{pnjBKZ}}(\beta))$;
7 $\quad \mathsf{GB} \leftarrow \mathsf{GB} + \mathsf{GB}_{\text{cum}} \cdot (1 - p_{\text{tot}}) \cdot P(\beta)$;
8 $\quad p_{\text{tot}} \leftarrow p_{\text{tot}} + (1 - p_{\text{tot}}) \cdot P(\beta)$;
9 $\quad$ **if** $p_{\text{tot}} > P_{\text{success}}$ **then**
10 $\quad\quad$ break;
11 **return** GB;

Algorithm 8: Expected Cost Version of [33].

when we running the last pump in the search step. See Equation 14 and Equation 16 for more detail.

In fact, the main difference between our Two-step-LWE-estimator and leaky-LWE-Estimator is the calculation of gates count of searching step G_2 since our estimator considers a two-step attack rather than reduction only mode (using BKZ class algorithm do reduction only to find the target vector). See Equation 16 for more detail.

Gates count of reduction step In this part, we introduce how to count the numbers of Gates when we solved uSVP in the reduction step. After we calculate each $\Pr[\mathrm{E}_i^{(1)}]$ value for $i \in \{1, 2, ...\}$ by using Equation 12 in the reduction step, we can calculate the expected value of gates counts G_1 of reduction step. We evaluate the expected value of gates counts G_1 of reduction step by Equation 14, see line 7 of algorithm 10 for more details.

$$G_1 = \sum_{i=1}^{end} \left[\Pr[\mathrm{W}_{(\beta_i, J_i)}^{(1)}] \cdot \left(1 - \Pr[\mathrm{E}_{i-1}^{(1)}]\right)\right] \cdot \sum_{l=40}^{\beta_i} Gates_{count}(l - d4f(\mathbf{B})) \cdot \frac{d - \beta_i}{J_i} \quad (14)$$

Besides, the $d4f(\mathbf{B})$ represents the maximum d4f value under current lattice basis $\mathbf{B}$ which initially has been reduced by a LLL and then reduced by a series pnj-BKZ. We give a d4f estimation $d4f(\mathbf{B})$ based on the current quality of lattice basis:

$$\max_f : \|\pi_f(\mathrm{v})\| \leq \mathrm{GH}\left(L_{[f:d]}(\mathbf{B})\right) \cdot \sqrt{4/3} \quad (15)$$

As the quality of lattice basis $\mathbf{B}$ gradually improved, the maximum d4f value $d4f(\mathbf{B})$ also increased since the better quality of lattice basis we get, the bigger the value of $\mathrm{GH}\left(L_{[f:d]}(\mathbf{B})\right)$. Equation 15 is more accurate than that given in [23]. Since the d4f estimation given in [23] is based on an assumption that the current quality of lattice basis reaches at least BKZ-$d/2$ reduction. Here d is the dimension of the embedding lattice. However, when we estimate the reduction effort of progressive BKZ with gradually increased blocksize, the quality of lattice basis also gradually improved rather than achieving BKZ-$d/2$ reduced at the very beginning.

Here we use the pnj-BKZ simulator to simulate how the quality of lattice basis changes during the process of reduction by a series pnj-BKZ. Therefore, compared with the d4f estimation in [23] we know the current quality of lattice basis which can help us to give the more accurate estimation of d4f value.

Gates count of searching step In this part, we introduce how to calculate the numbers of Gates when we solved u-SVP in the searching step. Because when we solved u-SVP in the searching step, it meant that all the pnj-BKZ algorithms in the reduction step failed to find the target vector. Meanwhile, the pump which we called to find the target vector in the searching step, uses a progressive sieving to find the target vector. Thus, based on Equation 13 to calculate $\Pr\left[\mathrm{E}^{(2)}_{(d_{svp})}\right]$, we use Equation 16 to calculate the expected value of gates of the searching step, see line 12 of Algorithm 10 for more details.

$$G_2 = \sum_{i=50}^{d_{svp}} \Pr\left[\mathrm{E}^{(2)}_{(i)}\right] \cdot \left(1 - \Pr[\mathrm{E}^{(1)}_{(end)}]\right) \cdot \left[\sum_{l=40}^{\beta_{end}} Gates_{count}\left(l - d4f(\mathbf{B})\right) \cdot \frac{d-\beta_i}{J_i} + \sum_{j=50}^{i-1} Gates_{count}\left(j - d4f(j)\right)\right] \tag{16}$$

When considering the cost of solving u-SVP during the searching step, it means that all pnj-BKZ did in the reduction step failed to find the target vector. We calculate the $\Pr\left[\bigwedge_{j=1}^{end} \mathrm{F}^{(1)}_{(\beta_j, J_j)}\right]$ in Equation 16 to represent the probability of all pnj-BKZ did in the reduction step failed to find the target vector. Besides before starting the big dimension Pump in the searching step, the total time cost of solving the u-SVP in the searching step already contains the full time cost of all pnj-BKZs we run in the reduction step. Therefore, the total gate count of reduction step is $\sum_{l=40}^{\beta_{end}} Gates_{count}(l - d4f(\mathbf{B})) \cdot \frac{d-\beta_i}{J_i}$ and when the dimension of SVP Oracle we considered equals i, the gate count of searching step is $\sum_{i>1,j=50}^{i-1} Gates_{count}\left(j - d4f(j)\right)$. Besiedes, the $d4f(j)$ is calculated by Equation 2.4.

Finally the total gates count for Two-step mode of solving LWE $G := G_1 + G_2$.

Memory count of Two-step LWE concrete estimator The Memory count of Two-step LWE concrete estimator we set as the maximum memory used during reduction step and searching step as Eq. (17).

$$B = \mathrm{bits}_{\mathrm{count}}(\max\left\{max\left\{\beta_i \in \mathsf{S}\right\}, d_{svp} - d4f(\mathbf{B})\right\}) \tag{17}$$

The number of LWE instances chosen in Two-step LWE concrete estimator Given a set of LWE parameters $(n, m, q, \sigma_s, \sigma_e)$ waiting for hardness evaluation. Determining the number of LWE instances used to construct the embedding lattice basis also will affect the security estimation of LWE. As the analysis in section D, we know that by optimizing the choice of the number of LWE instances according to the blocksize and jump selection strategy, the total cost of solving LWE decreased at most 5%. More detail also can be seen in Appendix D. Therefore we choose the same method in [13] and leaky-lwe estimator [43] to initially choose the number of LWE instances to construct the embedding lattice basis.

Estimate Hardness of LWE in Two-step Mode In this part, we give the detailed pseudocode of our Two-step LWE concrete hardness estimator, see Algorithm 10 for more detail. See https://github.com/Summwer/lwe-estimator-with-pnjbkz.git for the code implementation of our Two-step LWE concrete hardness estimator.

input : n, m, q, χ, P_{success};
output: $\mathsf{GB}_{\min}$;
$\mathsf{GB}_{\min} \leftarrow (+\infty, +\infty)$; $\mathsf{GB} \leftarrow (0,0)$; $\mathsf{GB}_{\text{pre}} \leftarrow (0,0)$; $p_{\text{tot}} \leftarrow 0$;
$\mathsf{rr} \leftarrow$ GSA profile of an LLL reduced $\text{LWE}_{n,m,q,\chi}$ instance;
for β **from** β_0 **to** d **do**
  $\mathsf{rr} \leftarrow \mathsf{SimBKZ}(\mathsf{rr}, \beta, 1)$;
  $P(\beta) \leftarrow \Pr\left[x \leftarrow \chi_\beta \middle| x \leq \mathsf{rr}_{[d-d_{\text{svp}}:d]}\right]$;
  $\mathsf{GB}_{\text{cum}} \leftarrow (\sum_{b=\beta_0}^{\beta} \mathsf{G}_{\text{pnjBKZ}}(b), \mathsf{B}_{\text{pnjBKZ}}(\beta))$;
  $\mathsf{GB}_{\text{pre}} \leftarrow \mathsf{GB}_{\text{pre}} + \mathsf{GB}_{\text{cum}} \cdot (1 - p_{\text{tot}}) \cdot P(\beta)$;
  $p_{\text{tot}} \leftarrow p_{\text{tot}} + (1 - p_{\text{tot}}) \cdot P(\beta)$; $\mathsf{GB}_{\text{csieve}} \leftarrow (0,0)$; $p_{\text{ctot}} \leftarrow p_{\text{tot}}$;
  for $d_{\text{svp}} \leftarrow \beta_0$ **to** d **do**
    $P(d_{\text{svp}}) \leftarrow \Pr\left[x \leftarrow \chi_{d_{\text{svp}}} \middle| x \leq \text{GH}(\mathsf{rr}_{[d-d_{\text{svp}}:d]})\right]$;
    $\mathsf{GB}_{\text{cum}} \leftarrow (\mathsf{GB}_{\text{cum}}[0] + \sum_{b=\beta+1}^{d_{\text{svp}}} \mathsf{G}_{\text{sieve}}(b), \mathsf{B}_{\text{sieve}}(d_{\text{svp}}))$;
    $\mathsf{GB}_{\text{csieve}} \leftarrow \mathsf{GB}_{\text{csieve}} + \mathsf{GB}_{\text{cum}} \cdot (1 - p_{\text{ctot}}) \cdot P(d_{\text{svp}})$;
    $p_{\text{ctot}} \leftarrow p_{\text{ctot}} + (1 - p_{\text{ctot}}) \cdot P(d_{\text{svp}})$;
    if $p_{\text{ctot}} > P_{\text{success}}$ **then**
      break;
  $\mathsf{GB} \leftarrow \mathsf{GB}_{\text{pre}} + \mathsf{GB}_{\text{csieve}}$;
  if $\mathsf{GB}[0] < \mathsf{GB}_{\min}[0]$ **then**
    $\mathsf{GB}_{\min} \leftarrow \mathsf{GB}$;
return $\mathsf{GB}_{\min}$;

Algorithm 9: Two-step LWE estimator with cum_prob

Consider the dimension-for-free situation for the estimator, the estimator is described as Algorithm 10.

E.2 Experiments on verifying the accuracy of Two-step LWE estimator

In this section, we mainly focus on the success probabilities of solving LWE by Two-step mode, especially the success probabilities of the last pump when using different sieving dimensions. we give the detail of our verification experiments to verify the Heuristic 7 which Equation 13 based on and the accuracy of Equation 13.

In particular, we use different parameters of the LWE instances to test the success probabilities of the last pump when using different progressive sieving dimensions. We choose four different LWE parameters $(n = 40, q = 1601, m = 1600, \alpha = 0.005)$, $(n = 40, q = 1601, m = 1600, \alpha = 0.015)$, $(n = 55, q = 3037, m = 3025, \alpha = 0.005)$, $(n = 45, q = 2027, m = 2025, \alpha = 0.010)$ to test. For each LWE parameter, we initial

input : n, m, q, χ, S;
output: $\mathsf{GB}_{\min}$;

Function `TwoStepLWEEsimator`$(n, m, q, \chi,\ \mathsf{S})$**:**
- $\mathsf{GB}_{\min} \leftarrow (+\infty, +\infty)$; $\mathsf{GB} \leftarrow (0,0)$; $\mathsf{GB}_{\mathrm{pre}} \leftarrow (0,0)$; $p_{\mathrm{tot}} \leftarrow 0$;
- $\mathsf{rr} \leftarrow$ GSA profile of an LLL reduced $\mathrm{LWE}_{n,m,q,\chi}$ instance;
- **for** $\beta \in \mathsf{S}$ *or* $(\beta, J) \in \mathsf{S}$ **do**
  - $\mathsf{rr} \leftarrow \mathsf{BKZSim}(\mathsf{rr}, \beta)$; // $\mathsf{PnjBKZSim}(\mathsf{rr}, \beta, J)$ if $J > 1$;
  - $P_{\mathrm{suc}}(\beta) \leftarrow \Pr\left[x \leftarrow \chi^2_\beta \middle| x \le \mathsf{rr}_{[d-\beta:d]}\right]$;
  - $\mathsf{GB}_{\mathrm{cum}} \leftarrow (\sum_{b=\beta_0}^{\beta} \mathtt{pbgate}(b - \mathtt{d4f}(b)), \mathtt{bit}(\beta - \mathtt{d4f}(\beta)))$;
  - $\mathsf{GB}_{\mathrm{pre}} \leftarrow \mathsf{GB}_{\mathrm{pre}} + \mathsf{GB}_{\mathrm{cum}} \cdot (1 - p_{\mathrm{tot}}) \cdot P(\beta)$;
  - $p_{\mathrm{tot}} \leftarrow p_{\mathrm{tot}} + (1 - p_{\mathrm{tot}}) \cdot P(\beta)$; $\mathsf{GB}_{\mathrm{csieve}} \leftarrow (0,0)$; $P_{\mathrm{suc}}(\beta_0 - 1) \leftarrow 0$;
  - **for** $d_{\mathrm{svp}} \leftarrow d_{\mathrm{start}}$ **to** d **do**
    - $P_{\mathrm{suc}}(d_{\mathrm{svp}}) \leftarrow \Pr\left[x \leftarrow \chi^2_{d_{\mathrm{svp}}} \middle| x \le (\mathrm{GH}(\mathsf{rr}_{[d-d_{\mathrm{svp}}:d]}))^2\right]$;
    - $\mathsf{GB}_{\mathrm{cum}}[0] \leftarrow \mathsf{GB}_{\mathrm{cum}}[0] + \mathtt{pgate}(d_{\mathrm{svp}} - \mathtt{d4f}(d_{\mathrm{svp}}))$;
    - $\mathsf{GB}_{\mathrm{cum}}[1] \leftarrow \max\{\mathsf{GB}_{\mathrm{cum}}[1], \mathtt{bit}(d_{\mathrm{svp}} - \mathtt{d4f}(d_{\mathrm{svp}}))\}$;
    - $\mathsf{GB}_{\mathrm{csieve}} \leftarrow \mathsf{GB}_{\mathrm{csieve}} + \mathsf{GB}_{\mathrm{cum}} \cdot (1 - p_{\mathrm{tot}}) \cdot (P(d_{\mathrm{svp}}) - P(d_{\mathrm{svp}-1}))$;
    - **if** $p_{\mathrm{tot}} + (1 - p_{\mathrm{tot}}) \cdot P(d_{\mathrm{svp}}) \ge 0.999$ **then**
      - break;
  - $\mathsf{GB} \leftarrow \mathsf{GB}_{\mathrm{pre}} + \mathsf{GB}_{\mathrm{csieve}}$;
  - **if** $\mathsf{GB}[0] < \mathsf{GB}_{\min}[0]$ **then**
    - $\mathsf{GB}_{\min} \leftarrow \mathsf{GB}$;
- **return** $\mathsf{GB}_{\min}$;

Algorithm 10: Two-step LWE Estimator

100 random LWE instances to construct 100 different lattice bases. Each lattice basis corresponds to an uSVP instance with a different target vector. Then we use pnj-BKZ to do pre-processing by some well-chosen reduction strategy S. Using LWE parameter $(n = 40, q = 1601, m = 1600, \alpha = 0.005)$ for example, we set $S = \{(\beta_1 = 10, J_1 = 1), ..., (\beta_{end} = 17, J_{end} = 1)\}$. Here, 100 different lattice basis under the same LWE parameter is used to simulate the situation that the distribution of the error vector of the LWE instance.

After pre-processing, we set the key parameter in the Pump of solving LWE, the value of κ. $\kappa \in \{0, ..., d\}$ will decide the size of the SVP we need to solve. In [13] they consider that one can solve an LWE by solving a $d - \kappa$ dimension SVP on $L_{[\kappa,d]}$ as long as $\sigma\sqrt{d-\kappa} < GH(L_{[\kappa,d]})$. Here $\sigma\sqrt{d-\kappa}$ is the expected value of the target vector. However, since we consider the square sum of the length of the target vector as a chi-squared distribution with $d - \kappa$ degrees of freedom, we calculate the cumulative probability of solving LWE when using a high-dimension Pump in section E.1 by Equation 13, and the line 12 of of Algorithm 10. To verify the Heuristic 7 Equation 13 based on and the accuracy of Equation 13, we actual test the success rate of solving LWE by Pump with different κ value.

More precisely, we set the d_{svp} in Pump from 30 to d by adjusting the value of κ and use each Pump with different d_{svp} value to try to find the solution of LWE on 100 different lattice basis which is reduced by pre-processing. Meanwhile, we record the actual success rate of each Pump with different d_{svp} values on 100 different lattice bases. Finally, we compare the actual success rate of each Pump with different d_{svp} with our estimation success rate of solving LWE by Pump in Equation 13, and the line 12 of Algorithm 10. See Figure 13 for more detail.

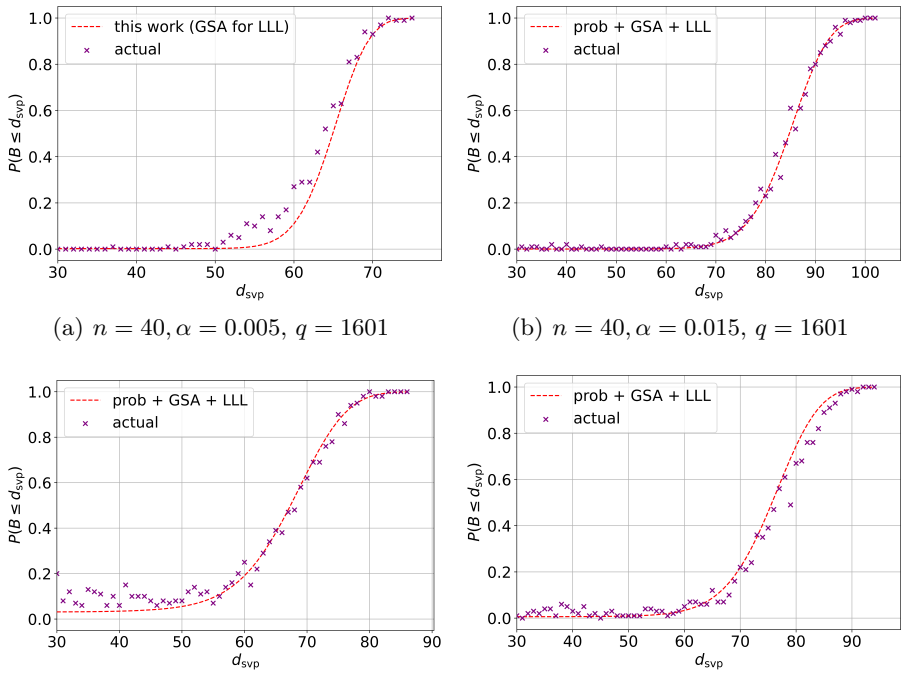
From Figure 13 we can see that the predication of the success rate of solving LWE given by Eq. (13), and the line 12 of Algorithm 10 is consistent with the experimental results, which means our analysis and estimation in Section E.1 is accurate.

E.3 The comparison of different estimation modes

In this part, we compare the Two-Step mode estimator using different reduction strategies (and considering whether to use the technology of list-decoding or not) with the leaky-lwe estimator.

We draw a Figure 14 to describe the relationship between the success rate of solving LWE estimated by different estimators and corresponding the expected number of gates. The blue line in Figure 14 is the relationship between the expected gates count and the accumulation success probability of solving LWE by pure progressive BKZ with trivial reduction strategy S_0. These Two-step lines in Figure 14 are the relationship between the expected gates count and the accumulation success probability of solving LWE by two-step mode whose reduction step also used a trivial reduction strategy $S_1 : S_1 \subsetneqq S_0$ (In two-step mode it will end the reduction step at a suitable time to make the total cost minimal). These Enumbs lines in Figure 14 also are the relationship between the expected gates count and the accumulation success probability of solving LWE by two-step mode while the reduction strategy is the optimized blocksize and jump selection strategy. The [AGPS20] means estimation does not use the list-decoding technology proposed in [32], while [MATZOV22] means using the list-decoding technology [32] to evaluate the security strength.

From Figure 14 we can see that no matter the LWE challenge instances or the LWE instances in NIST candidates, the accumulation success probability of solving LWE

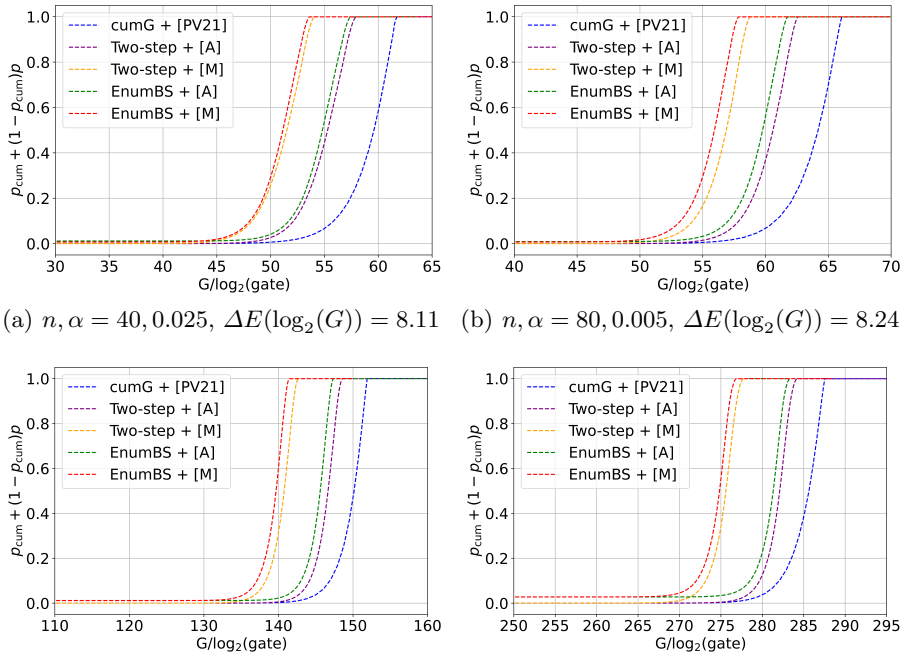


(a) $n = 40, \alpha = 0.005,\ q = 1601$ (b) $n = 40, \alpha = 0.015,\ q = 1601$

(c) $n = 55, \alpha = 0.005,\ q = 2501$ (d) $n = 45, \alpha = 0.010,\ q = 2027$

Fig. 13. Verification experiments of Equation 13 which calculates the success probability of solving LWE by Pump. See the line 12 of of Algorithm 10 for more details.

by two-step mode is much faster approach 1 than that of the leaky-lwe estimator. Besides the expected number of gates in the two-step solving mode is smaller than that of the leaky-lwe estimator when the accumulation success probability of solving LWE approaches 1. The evaluation result shows that the leaky-lwe estimator gives an optimistic estimation. In addition, both the optimized blocksize and jump selection strategy and the list-decoding technology proposed in [32] can further decrease the estimated security strength by replacing the trivial reduction strategy in Two-step mode. See Figure 14 or Table 9 for more details about the difference between different estimation modes.



(a) $n, \alpha = 40, 0.025$, $\Delta E(\log_2(G)) = 8.11$ (b) $n, \alpha = 80, 0.005$, $\Delta E(\log_2(G)) = 8.24$

(c) Kyber512, $\Delta E(\log_2(G)) = 10.48$ (d) Kyber1024, $\Delta E(\log_2(G)) = 10.68$

Fig. 14. Comparison between the output of Cumulated Cost Version of [33](Algorithm 8) and Two-step mode(Algorithm 10, this work) for lwe challenge $(n, \alpha) \in \{(40, 0.025), (80, 0.005)\}$ and on Kyber 512 and 1024 [2]. The difference in predicted mean first viable block size between the two simulators is reported as $\Delta E(\log_2(G))$.

F The Optimized Strategy for the LWE Challenge

In Table 10, we give the optimized blocksize and jump strategy generated by EnumBS for solving TU Darmstadt LWE Challenge instances with

$$(n, \alpha) \in \{(40, 0.035), (40, 0.040), (50, 0.025), (55, 0.020), (90, 0.005)\}$$

Table 9. Security bits decreasing ($\Delta E(\log_2(G))$) under different estimation modes

	(n=40,α=0.025)	(n=80,α=0.005)	Kyber512	Kyber1024
Two-Step with Trivial Strategy [AGPS20]	3.78	3.51	3.34	3.34
Two-Step with Trivial Strategy [MATZOV22]	7.68	7.33	9.29	9.83
Two-Step with EnumBS [AGPS20]	4.29	4.34	4.55	4.22
Two-Step with EnumBS [MATZOV22]	8.11	8.24	10.48	10.68

successfully.

Table 10. Blocksize and Jump strategy generated by EnumBS.

(n, α)	Strategy (β, jump)	EnumBSGen/s
(40,0.035)	[(76,8),(91,9),(117,10),(117,4),(132,4)]	1700
(40,0.040)	[(73,8),(117,10),(117,10),(119,4),(135,4),(147,4)]	2800
(50,0.025)	[(76,8),(91,9),(114,10),(117,10),(117,4), (128,4),(141,4),(147,4)]	5200
(55,0.020)	[(94,9),(117,10),(117,10),(117,4),(128,4), (132,4),(141,4),(147,4)]	6500
(90,0.005)	[(89,9),(92,9),(116,10),(116,10),(117,10),(117,10), (117,4),(118,4),(121,4),(128,4),(132,4),(138,4),(141,4)]	12000

G Impact on the NIST lattice based scheme

In this section, we quantitative analysis of the impact of the Two-step LWE solving mode and optimized blocksize and jump selection strategy on NIST PQC schemes.

We change the gate count of list-decoding from [30] to [32] in the estimation. By using the list-decoding technology proposed in [32] it will further decreasing the estimated security strenth of NIST PQC candidates.

The evaluation results show that combining our optimized blocksize and jump selection strategy and Two-step mode with the new gate count of the list-decoding method [32] can further reduce the estimated security bits of LWE. Specifically, under the RAM model, the estimated security bit of LWE in NIST schemes [28] can be reduced by 8.0∼10.7 bit. Besides, in Table 11 even though ΔB is negative (our strategies use bigger memory). But since the time cost (Number of gates G) is far larger than than the memory cost B, the impact of memory growth can be ignored with such a significant decreasing in time cost. Specifically, in Table 11 even though ΔB is negative (Two-step mode use bigger memory), $\Delta \log_2(\mathrm{G} + \mathrm{B})$ is same as $\Delta \log_2 \mathrm{G}$ between 8.0∼10.7 bit using list-decoding skill proposed in [32]. It means that the increase in memory will partially offset the decrease in the number of gates. However, in general, the time cost is still decreasing even considering the extra increase of memory. See Table 11 for details.

Here G and B in Table 11 respectively represent the total log number of logic circuits for event W happened and the maximum memory needed for event W happened.

H Pseucode for BSSA

The concrete process for BSSA is as Alg. 11.

Table 11. Security Estimation results of different estimator for NIST schemes[♮].

	$\log_2$G/log2(gate)			$\log_2$B/log2(bit)			ΔG
	Previous	Two-Step	EnumBS	Previous	Two-Step	EnumBS	
Kyber512	151.5	142.6	141.4	93.8	99.1	98.1	10.1
Kyber768	215.1	204.4	196.3	138.5	143.2	137.3	10.7
Kyber1024	287.3	277.7	276.9	189.7	195.4	194.6	10.4
Dilithium-II	158.6	150.8	150.6	97.8	104.3	104.4	8.0
Dilithium-III	216.7	207.9	207.9	138.7	145.3	145.3	8.8
Dilithium-V	285.4	277.0	277.0	187.4	194.1	194.1	8.4

♮ In the column of G/log2(gates) and B/log2(bit), respectively reflect the security bit estimations and memory bit estimations of LWE instances in NIST schemes with different estimators. Here under the influence of the optimized blocksize and jump selection and two-step mode strategy, ΔG means the security bits decreasing under the RAM model and ΔG+ΔB means the security bits decreasing by considering the sum of time cost and memory cost. MATZOV presented a new list-decoding method in 2022 [32] and Martin Albrecht *et al* estimates its gate count in his LWE estimator(https://github.com/malb/lattice-estimator), so we apply the new gate count of list-decoding mehthod for the estimation on the NIST standardization.

input : rr_0, $F(\star, \mathcal{D})$, $\beta^{\text{start}} \leftarrow 50$, $J_{\max}(\star) \leftarrow \text{d4f}(\star)/2$;
output: $T_{\min}$, $S_{\min}$;
Function BSSA(rr_0, $F(\star, \mathcal{D})$, $\beta^{\text{start}} \leftarrow 50$, $J_{\max}(\star) \leftarrow \text{d4f}(\star)/2$):
  $d \leftarrow \text{len}(\mathsf{rr}_0)$; $\text{PSC}^{(0)} \leftarrow$ ProSieveDimEst(rr_0, $F(\star, \mathcal{D})$); $\mathsf{BS}[\beta^{\text{start}}] = (\mathsf{rr}_0, [\,], 0, \text{PSC}^{(0)})$;
  for $\beta \leftarrow \beta^{\text{start}}$ **to** d **do**
    $T^{(\min)}_{\text{PnjBKZs}} \leftarrow +\infty$;
    for $\beta^{\text{sstart}} \leftarrow \beta^{\text{start}}$ **to** $\beta - 1$ **do**
      $\mathsf{bs}^{\text{sstart}} \leftarrow \mathsf{BS}\,[\beta^{\text{sstart}}]$; $\mathsf{bs} \leftarrow (\emptyset, \emptyset, +\infty, +\infty)$;
      Update bs^* under strategy $\mathsf{bs}^{\text{sstart}}.\mathsf{S} \cup [(\beta, 1, \sharp\text{tours(BKZ-}\beta)]$;
      for $\beta^{\text{alg}} \leftarrow \beta + 1$ **to** d **do**
        for $j \leftarrow J_{\max}(\beta^{\text{alg}})$ **to** 1 **do**
          $T' \leftarrow +\infty$;
          for $t \leftarrow 1$ **to** $\sharp\text{tours(PnjBKZ-}(\beta^{\text{alg}}, j))$ **do**
            Update bs' under strategy $\mathsf{bs}^{\text{sstart}}.\mathsf{S} \cup [(\beta^{\text{alg}}, j, t)]$;
            if $\mathsf{bs}'.\text{PSC} < \mathsf{bs}^*.\text{PSC}$ **then**
              $T' \leftarrow \mathsf{bs}'.T_{\text{PnjBKZs}}$;
              break;
          if $\mathsf{bs}.T_{\text{PnjBKZs}} > T'$ **then**
            $\mathsf{bs} \leftarrow \mathsf{bs}'$;
      if $T^{(\min)}_{\text{PnjBKZs}} > \mathsf{bs}.T_{\text{PnjBKZs}}$ **then**
        $T^{(\min)}_{\text{PnjBKZs}} \leftarrow \mathsf{bs}.T_{\text{PnjBKZs}}$; $\mathsf{BS}\,[\beta] \leftarrow \mathsf{bs}$;
  $\mathsf{bs}_{\min} \leftarrow \min_{\mathsf{bs}.T_{\text{PnjBKZs}} + \mathsf{bs}.\text{PSC}} \mathsf{BS}$;
  return $T_{\min} \leftarrow \mathsf{bs}.T_{\text{PnjBKZs}} + \mathsf{bs}.\text{PSC}$, $\mathsf{S}_{\min} \leftarrow \mathsf{bs}_{\min}.\mathsf{S}$;

Algorithm 11: BSSA