I. J. Mathematical Sciences and Computing, **2022, 1, 1-17**
Published Online February 2022 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmsc.2022.01.01

Optimal bounding function for GNR-enumeration

Gholam Reza Moghissi
Department of ICT, Malek-Ashtar University of Technology, Tehran, Iran
E-mail: fumoghissi@chmail.ir

Ali Payandeh
Department of ICT, Malek-Ashtar University of Technology, Tehran, Iran

Received: 14 September 2021; Revised: 21 October 2021; Accepted: 21 November 2021; Published: 08 February 2022

Abstract: The proposed pruning technique by Gama-Nguyen-Regev for enumeration function makes this pruned enumeration (GNR-enumeration) as a claimant practical solver for SVP. The total cost of GNR-enumeration over a specific input lattice block with pre-defined enumeration radius and success probability would be minimized, just if this enumeration uses an optimal bounding function for pruning. Unfortunately, the running time of the original proposed algorithm of searching optimal bounding function by the work of Chen-Nguyen (in 2011) is not analyzed at all, so our work in this paper tries to introduce some efficient searching algorithms with exact analysis of their time/space complexity. In fact, this paper proposes a global search algorithm to generate the optimal bounding function by a greedy idea. Then, by using our greedy strategy and defining the searching steps based on success probability, a practical search algorithm is introduced, while it's time-complexity can be determined accurately. Main superiorities of our algorithm include: complexity analysis, using high-performance version of each sub-function in designing search algorithm, jumping from local optimums, simple heuristics to guide the search, trade-off between quality of output and running time by tuning parameters. Also by using the building blocks in our practical search algorithm, a high-quality and fast algorithm is designed to approximate the optimal bounding function.

Index Terms: Search algorithm, Approximate algorithm, Success probability, Enumeration cost, Time complexity.

1. Introduction

Lattice-based cryptography is one of the main candidates in post-quantum cryptography [1]. The security of lattice based cryptography comes from the hardness of the lattice problems. Since hardness of Shortest Vector Problem (SVP) affects security aspects of many lattice-based cryptographic schemes, so the study of attacks on this problem (SVP) is valuable. Some solvers of (approximate) SVP include: lattice basis reduction, Voronoi-cell, sieving, random sampling, evolutionary search [2], etc. Lattice enumeration is one of the main algorithms in theory and practice for solving SVP within the polynomial space (memory). However new achievements in lattice enumeration are notable (such as the works of [3] as new enumeration by integrating sparse orthogonalized integer representations for shortest vectors and [4] as a quantum version of lattice enumeration), this paper just focuses on Gama-Nguyen-Regev (GNR) enumeration proposed in [5].

The running-time of enumeration function is affected by preprocessing the lattice block and using the effective pruning techniques [5]. Bounding function vector effectively prunes GNR-enumeration tree [5]. An optimal bounding function is defined as a bounding function which prunes most number of nodes in GNR-enumeration over an input lattice block with specific success probability of finding solution. Chen and Nguyen propose a generator for optimal bounding function in Algorithm 5 of [6]. However, the quality of generated bounding functions in Chen-Nguyen's test results are acceptable, the time-complexity of their algorithm is not analyzed and it's termination condition is set just by hand (as 1 hour), not by analysis of search convergence. It is clear that complexity analysis is necessary in practical block sizes for theoretical/practical purpose (e.g., the research of [7] or our idea on progressive-BKZ in [8]).

We define an approximation in Claim 2 from [9] for estimating the cost of GNR-enumeration pruned by optimal bounding function. Also, we show in [10], if the success probability of optimal bounding function is closed to %100, then the cost of GNR-enumeration pruned by this bounding function can be approximated as $\frac{1}{2^{\beta/4}}$ times the cost of full-enumeration (which verifies the analysis in [5]). In other side, we show in [10], for practical block size of $100 \leq \beta \leq 250$, the upper-bound for speedup of GNR-enumeration pruned by a piecewise-linear bounding function with much small success probability (as extreme pruning) when the lattice block is preprocessed by BKZ with enumeration (as

SVP-solver) would be estimated from $2^{\frac{\beta}{6.6}}$ to $2^{\frac{\beta}{4.4}}$ (and tries to reject the claimed speedup of $2^{\frac{\beta}{2}}$ by extreme-pruning over full-enumeration in [5]). In fact, no ways are suggested in [9,10] to generate optimal bounding function.

Unfortunately, the running time of the original proposed algorithm of searching optimal bounding function by the work of Chen-Nguyen in [6] is not analyzed at all. In fact, this paper proposes more efficient generators of optimal bounding function to minimizing the cost (running time) of GNR-enumeration, while the time/space complexity of our generators are analyzed accurately. To the best of our knowledge, analyzing and improving the original generator of optimal bounding function in [6] are not discussed in former studies.

In fact, in this paper, a global search of optimal bounding function (Algorithm 1) is introduced by a greedy idea. Because of unsuitable steps in this search algorithm (Algorithm 1), it's running-time is not tolerable for practical purpose! Moreover, it's time-complexity cannot be analyzed accurately! Re-defining the searching steps based on success probability of bounding functions (not based on the difference of entries of bounding functions in Algorithm 1) leads to possibility of time-complexity analysis. By using the greedy strategy in our global search (Algorithm 1) together with the searching steps based on success probability, this paper introduces another new search algorithm (Algorithm 2). However this search algorithm (Algorithm 2) is not a global search, it includes some techniques to jump from local optimums and some simple heuristics which guide the search toward the global optimum. Also our new search function (Algorithm 2) uses various useful parameters which can be tuned to make trade-off between quality of output and running time. Besides, all building blocks in the design of this algorithm use the best algorithmic techniques by the authors for better speedup. Finally, by using the building blocks of this search algorithm (Algorithm 2), an efficient algorithm (Algorithm 3) is introduced to approximate the optimal bounding function with acceptable quality (i.e., these generated bounding functions introduce the enumeration cost which is closed to optimal enumeration cost). Time-complexity of our new search algorithm (Algorithm 2) and approximate algorithm (Algorithm 3) are analyzed accurately and needed test results for justifying the value of them are introduced in this paper.

This paper is organized as follows. In Section 2, the background information is described. Our contributions are introduced in Section 3. Section 4 is dedicated to complexity analysis of our proposed algorithms. Also our test results are presented in Section 5. Finally, the conclusion of this work is declared in Section 6.

2. Preliminaries

In this section, a sufficient background is introduced to make this work easy to be understood. A lattice is defined by n-linearly independent vectors $b_1, \dots, b_n \in \mathbb{R}^m$, as following set of n-dimensional points:

$$\mathcal{L}(b_1, \dots, b_n) = \{\textstyle\sum_{i=1}^{n} x_i b_i : x_i \in \mathbb{Z}\}. \tag{1}$$

The set of $B = [b_1, \dots, b_n]$ is known as lattice basis of lattice. For cryptographic application, we only consider integer lattices by setting $b_i \in \mathbb{Z}^m$. The rank and dimension of lattice $\mathcal{L}(B)$ are referred respectively by n and m.

The concepts of Gram-Schmidt Orthogonalization and projected sub-lattices are other fundamental definitions. For a given lattice basis $B = [b_1, b_2, \dots, b_n]$, the orthogonal projection $\pi_i(\dots)$ is defined as follows:

$$\pi_i: \mathbb{R}^m \mapsto \operatorname{span}(b_1, \dots, b_{i-1})^{\perp}, \text{ where } 1 \le i \le n. \tag{2}$$

For given basis $B = [b_1, \dots, b_n]$, the Gram-Schmidt orthogonal (GSO) basis $B^* = [b_1^*, \dots, b_n^*]$ is defined as follows:

$$\pi_i(b_i) = b_i^* = b_i - \textstyle\sum_{j=1}^{i-1} \mu_{i,j}. b_j^*, \quad \text{where } \mu_{i,j} = \frac{b_i . b_j^*}{\|b_j^*\|^2} \text{ and } 1 \le j < i \le n. \tag{3}$$

In this paper, we mostly focus on the lattice blocks of an input lattice basis. In fact, a lattice block from basis B, from vector index j to k with block size of $\beta = k - j + 1$ is defined as $B_{[j,k]} = [b_j, b_{j+1}, \dots, b_k]$. Also, projected lattice block $\mathcal{L}_i$ in this paper is defined as follows:

$$\mathcal{L}_i = \mathcal{L}[\pi_{\beta-i+1}(b_{\beta-i+1}), \dots, \pi_{\beta-i+1}(b_\beta)]. \tag{4}$$

One of the main heuristic in lattice theory is Gaussian Heuristic, which estimates the number of lattice vectors in a measurable space S [5]. The expected norm of shortest vector in lattice $\mathcal{L}$ can be estimated by using Gaussian Heuristic of lattice $\mathcal{L}$ as follows:

$$\mathrm{GH}(\mathcal{L}) \approx \sqrt{\frac{n}{2\pi e}}(\det B)^{\frac{1}{n}}. \tag{5}$$

Lattice enumeration is one of the main functions of BKZ algorithm, while it can be used as an independent solver of SVP too. This paper focuses only on GNR-pruned type of enumeration which is defined based on bounding function concept. A bounding function is a pruning method for enumeration which is defined by a vector of $\mathcal{R} = [\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_\beta]$, where $0 \le \mathcal{R}_1 \le \mathcal{R}_2 \le \cdots \le \mathcal{R}_\beta = 1$ (see [5]). By using bounding function, for lattice block $\mathcal{L}(b_j, \dots, b_k)$ and coefficient vector $x \in \mathbb{Z}^\beta$, GNR-pruning replaces inequality of $\|\pi_{k+1-i}(x.B_{[j,k]})\| \le R$ for $1 \le i \le k-j+1$ as a bounded ball (in full-enumeration) by $\|\pi_{k+1-i}(x.B_{[j,k]})\| \le \mathcal{R}_i R$, as a cylinder-intersection [6], where $0 \le \mathcal{R}_1 \le \cdots \le \mathcal{R}_{k-j+1}$=1. The pseudo-code of GNR-pruned enumeration can be studied in Appendix B from [5].

For lattice block of $\mathcal{L}(b_j, \dots, b_k)$, initial enumeration radius R and bounding function $\mathcal{R}$, if there is just one lattice vector v in n-dimensional ball with radius of R (i.e., $\|v\| \le R$), the probability of finding v after GNR-pruning by $\mathcal{R}$ in enumeration tree is defined as success probability of $\mathcal{R}$, which is shown by $p_{succ}(\mathcal{R})$ [5]. However success probability of bounding function can be estimated by Monte Carlo simulation (see Algorithm 6 in [6]), this is not efficient enough [5], so this paper uses Algorithm 7 from [6] as an efficient way of this estimating. By definition of p_{succ} as success probability, for lattice block $\mathcal{L}_\beta$ with size of β, the total nodes of GNR-enumeration with a bounding function $\mathcal{R}$ and initial radius R over this block can be estimated by (6) [5]:

$$N(\mathcal{L}_\beta, \mathcal{R}, R) = \frac{1}{2}\sum_{i=1}^{\beta}\left(\frac{\mathcal{R}_i R}{\mathrm{GH}(\mathcal{L}_i)}\right)^i p_{succ}(\mathcal{L}_i, \mathcal{R}_{[1\dots i]}, R). \tag{6}$$

Note: In this paper, the expressions of “total nodes”, “number of tree nodes” and “cost” have the same meaning.

Accordingly, the formal definition of optimal bounding function can be declared as follows [9]:

Optimal bounding function. For input lattice block $\mathcal{L}_{[j,k]}$ and the enumeration radius $R \approx \|v\|$ where v is expected to be the final solution vector of GNR-enumeration with an input success probability P, the optimal bounding function $\mathcal{R}_{\mathrm{opt}}$ with success probability P can be defined formally as following set [9]:

$$\mathcal{R}_{\mathrm{opt}} \in \{\mathcal{R} \mid p_{succ}(\mathcal{R}) = \mathrm{P}\ \&\ \forall\, \mathcal{R}' : N(\mathcal{L}_{[j,k]}, \mathcal{R}, R) \le N(\mathcal{L}_{[j,k]}, \mathcal{R}', R)\}, \tag{7}$$

where $R \approx \|v\|$.

As mentioned in Section 1, one of the main parts of this paper focuses on complexity analysis of generating optimal bounding function. The main input parameter in our complexity analysis is block size of β. Since the parameter of β in practical running of GNR-enumeration cannot be any large size, this paper doesn’t use the regular asymptotic notations in complexity analysis, such as Big O “O”, Big Omega “Ω”, Theta “θ”, etc.; Instead, by assuming the range of $50 < \beta < 400$, this paper just tries to estimate the complexity function of each algorithm, based on some basic operations in that algorithm. For simplicity in analysis, the time complexity of these basic operations is approximated by a factor of $\mathcal{C} \approx O(1)$ for all algorithms. The complexity function of each algorithm is shown by the notation of T_i in this paper.

3. Our Proposed Techniques for Generating Optimal Bounding Function

In this section, our global search idea of optimal bounding function is introduced. Next, our practical search algorithm is introduced. Finally, our approximate algorithm with acceptable time-complexity is proposed in this section.

3.1. Global Search of Optimal Bounding Function

Algorithm 1 is our global search algorithm by a greedy strategy to find the optimal bounding function.

Note: The notation of “0^+” in Algorithm 1 is assumed as the closet positive real number to zero, while “ε” is just a small positive real number.

```
Algorithm 1 Generate_$\mathcal{R}_{\text{opt}}$_1
Input: lattice block $\mathcal{L}_\beta$, enum radii $R$, success prob $\mathrm{P}_{\text{final}}$, error $\varepsilon$
1:  $\mathcal{R}_{[0,1,\dots,\beta]} = [0^+, 1,1,1,\dots,1]$; /*Full enumeration bounding function besides $\mathcal{R}_0 = 0^+$ for correctness of algorithm*/
2:  while($\exists\, 1 \le i \le \beta, \forall\, 1 \le j \le \beta$: $\frac{N(\mathcal{L}_\beta,[\mathcal{R}_1\dots\mathcal{R}_\beta],R)-N(\mathcal{L}_\beta,[\mathcal{R}_1\dots\mathcal{R}_i-\varepsilon\dots\mathcal{R}_\beta],R)}{p_{succ}(\mathcal{L}_\beta,[\mathcal{R}_1\dots\mathcal{R}_\beta],R)-p_{succ}(\mathcal{L}_\beta,[\mathcal{R}_1\dots\mathcal{R}_i-\varepsilon\dots\mathcal{R}_\beta],R)} < \frac{N(\mathcal{L}_\beta,[\mathcal{R}_1\dots\mathcal{R}_\beta],R)-N(\mathcal{L}_\beta,[\mathcal{R}_1\dots\mathcal{R}_j-\varepsilon\dots\mathcal{R}_\beta],R)}{p_{succ}(\mathcal{L}_\beta,[\mathcal{R}_1\dots\mathcal{R}_\beta],R)-p_{succ}(\mathcal{L}_\beta,[\mathcal{R}_1\dots\mathcal{R}_j-\varepsilon\dots\mathcal{R}_\beta],R)}$ &
            $p_{succ}(\mathcal{L}_\beta,[\mathcal{R}_1\dots\mathcal{R}_i-\varepsilon\dots\mathcal{R}_\beta],R) \ge \mathrm{P}_{\text{final}}$ & $p_{succ}(\mathcal{L}_\beta,[\mathcal{R}_1\dots\mathcal{R}_j-\varepsilon\dots\mathcal{R}_\beta],R) \ge \mathrm{P}_{\text{final}}$ & $0 < \mathcal{R}_{j-1} \le \mathcal{R}_j - \varepsilon$ & $0 < \mathcal{R}_{i-1} \le \mathcal{R}_i - \varepsilon$ & $i \ne j$){
3:      $\mathcal{R}_i \leftarrow \mathcal{R}_i - \varepsilon$;}//end while
Output: Optimal bounding function $\mathcal{R}_{opt} = \mathcal{R}$.
```

Conjecture 1 defines our main idea behind this algorithm:

Conjecture 1. For given lattice block $\mathcal{L}_\beta$, specific success probability P and initial radius R, the cost of GNR-enumeration pruned by bounding function $\mathcal{R}'$ generated in Algorithm 1 would be minimal when $\varepsilon \to 0^+$:

$$N(\mathcal{L}_\beta, \mathcal{R}', R) = \min_{\mathcal{R}: 0 \le \mathcal{R}_1 \le \dots \le \mathcal{R}_\beta = 1} N(\mathcal{L}_\beta, \mathcal{R}, R),$$

where $\mathcal{R}' \leftarrow \text{Generate\_}\mathcal{R}_{\text{opt}}\text{\_1}(\mathcal{L}_\beta, R, \mathrm{P}, \varepsilon)$ and $\varepsilon \to 0^+$. However there are infinite bounding function vectors for an input success probability P, Conjecture 1 implicitly claims that optimal bounding function for lattice $\mathcal{L}_\beta$, success probability P and initial radius R is unique. Moreover, this conjecture claims that optimal bounding functions of $\mathcal{R}$ and $\mathcal{R}'$, respectively for target success probabilities of P and P′ satisfies following condition:

$$\forall\, \mathcal{R} \leftarrow \text{Generate\_}\mathcal{R}_{\text{opt}}\text{\_1}(\mathcal{L}_\beta, R, \mathrm{P}, \varepsilon), \forall\, \mathcal{R}' \leftarrow \text{Generate\_}\mathcal{R}_{\text{opt}}\text{\_1}(\mathcal{L}_\beta, R, \mathrm{P}', \varepsilon): (\mathrm{P} < \mathrm{P}') \Rightarrow (\forall\, 1 \le i \le \beta: \mathcal{R}_i \le \mathcal{R}'_i). \tag{8}$$

The proposition (8) proposes that Algorithm 1 is used as a preprocess to generate a spool of optimal bounding functions with success probabilities from $\mathrm{P}_1 < \mathrm{P}_2 < \mathrm{P}_3 < \dots < \mathrm{P}_x$.

For practical purposes, this is not possible to use small step of ε, since time-complexity of algorithm cannot be tolerated! In other sides, big step of ε along with non-accurate estimators of N and p_{succ}, change Algorithm 1 as a global search into a local search which frequently sticks in local optimums!

3.2. Practical Search of Optimal Bounding Function

This paper introduces effective improvements on Algorithm 1 to make it efficient for practical purpose (see our practical search algorithm in Algorithm 2).

```
Algorithm 2 Generate_$\mathcal{R}_{\text{opt}}$_3
Input: lattice block $\mathcal{L}_\beta$, enum radii $R$, target success prob $\mathrm{P}_{\text{final}}$, $\text{Layer}_{\text{Count}}$, $\text{jump}_{\text{len}}$, $\text{jump}_{\text{width}}$, $\text{TypeOfBF}_{\text{top}}$, $\text{index}_{\text{start}}$.
1:   $\mathrm{P}_{\text{top}} = \min(0.9999, 2 \times \mathrm{P}_{\text{final}})$; $\mathrm{P}_{\text{low}} = \frac{\mathrm{P}_{\text{top}}}{\text{Layer}_{\text{Count}}+1}$;
2:   $\text{err}_{\text{layer}} = \frac{\mathrm{P}_{\text{low}}}{\beta}$; $\text{err}_{\text{column}} = \frac{\mathrm{P}_{\text{low}}}{\text{Layer}_{\text{Count}}+1}$;
3:   $\mathcal{R}_{\text{top}} \leftarrow \text{Generate\_}\mathcal{R}_{\text{type}}(\beta, \mathrm{P}_{\text{top}}, \text{TypeOfBF}_{\text{top}}, \text{err}_{\text{layer}})$;
4:   $\mathcal{R}_{\text{low}} \leftarrow \text{Generate\_}\mathcal{R}_{\text{low}}(\beta, \mathrm{P}_{\text{low}}, \mathrm{P}_{\text{top}}, \mathcal{R}_{\text{top}}, \text{err}_{\text{column}})$;
5:   $\ell_{\text{next}} = 1$; //next layer of steps to decrease entries of $\mathcal{R}$
6:   $\mathcal{R}_{\boldsymbol{\varepsilon}} \leftarrow \mathcal{R}_{\text{top}}$; //$\mathcal{R}_{\boldsymbol{\varepsilon}}[i] \coloneqq \sum_{l=1}^{\ell_{\text{next}}-1}(\mathcal{R}_{\text{top}}[i] - \boldsymbol{\varepsilon}[l][i])$
7:   $\boldsymbol{\varepsilon}[\ell_{\text{next}}][1\dots\beta] \leftarrow \text{Generate\_}\varepsilon_{\text{array}}(\beta, \mathrm{P}_{\text{low}}, \mathcal{R}_{\text{low}}, \mathrm{P}_{\text{top}}, \mathcal{R}_{\text{top}}, \mathcal{R}_{\boldsymbol{\varepsilon}}, \text{err}_{\text{layer}}, \ell_{\text{next}})$;
8:   $\ell_{\text{next}}++$;//next layer of steps is set for next call.
9:   $\mathcal{R}_{\text{initial}} \leftarrow \mathcal{R}_{\text{top}}$;//initial bounding function in searching
10:  Add_JumpingSteps($\beta, \mathcal{R}_{\text{initial}}, \text{jump}_{\text{len}}, \text{jump}_{\text{width}}, \text{index}_{\text{start}}\ \boldsymbol{\varepsilon}, \mathrm{P}_{\text{low}}, \mathcal{R}_{\text{low}}, \mathrm{P}_{\text{top}}, \mathcal{R}_{\boldsymbol{\varepsilon}}, \mathcal{R}_{\text{top}}, \text{err}_{\text{layer}}, \ell_{\text{next}}$);
11:  $\mathcal{R}_{opt} \leftarrow \text{Generate\_}\mathcal{R}_{\text{opt}}\text{\_2}(\mathcal{L}_\beta, \mathcal{R}_{\text{initial}}, \mathrm{P}_{\text{final}}, \boldsymbol{\varepsilon}, \ell_{\text{next}}, \mathcal{R}_{\text{initial}}, \text{Layer}_{\text{Count}})$;
Output: optimal bounding function $\mathcal{R}_{opt}$.
```

Our main improvements in Algorithm 2 (over Algorithm 1) can be counted as follows:

- Using equal steps based on success probability between the searched states (by matrix of $\boldsymbol{\varepsilon}$) makes it possible to analyze the time-complexity, while this is not possible for algorithm 1 (since searching step in Algorithm 1 is a constant of ε based on entries of bounding function).
- Making trade-off between quality of output bounding function and running time by tuning input parameter.
- Since total nodes N is a high-order function based on each entry $\mathcal{R}_i$, while theses entries are too limited as: $0 < \mathcal{R}_1 \le \dots \le \mathcal{R}_{i-1} \le \mathcal{R}_i \le \mathcal{R}_{i+1} \le \dots \le \mathcal{R}_\beta = 1$, so this is not possible to return an optimal bounding function by big constant step ε in Algorithm 1, therefore Conjecture 1 forces implicitly to use the condition of

$\frac{N(\text{State}_{\text{last}})-N(\text{State}_i)}{p_{succ}(\text{State}_{\text{last}})-p_{succ}(\text{State}_i)} < \frac{N(\text{State}_{\text{last}})-N(\text{State}_j)}{p_{succ}(\text{State}_{\text{last}})-p_{succ}(\text{State}_j)}$
so that $0 < \max_{i=1 \text{ to } \beta}[p_{succ}(\text{State}_{\text{last}}) - p_{succ}(\text{State}_i)] < \varepsilon_0$ (where ε_0 is a small threshold of probability). In other sides, small step ε leads to slow decreasing of first entries of bounding function, in compare with last entries of bounding function, and this leads to much running-time since last entries of bounding function are lower-bounded by their previous entries)! Besides, for non-exact total nodes N and success probability p_{succ}, the main condition in line 2 from Algorithm 1 can direct search wrongly! In Algorithm 2, equal searching step based on success probability (by matrix $\boldsymbol{\varepsilon}$) solves this problem.
- Algorithm 2 introduces some effective techniques to jump from local optimums (however abuse of jumping steps makes the search biased and even in-effective).
- Using suitable upper-bound for entries of searched bounding function can prune the search space in Algorithm 2 considerably, while this can guide searching process toward the global optimum as a heuristic (however bad shaped upper-bound or some ones with too fitted success probability makes the search biased).
- By efficient algorithmic techniques along with suitable data structures, Algorithm 2 introduces better speedup.

At the reminder of this sub-section, the design considerations of Algorithm 2 are discussed.

1) Function of $Generate\_\mathcal{R}_{type}$: By using this function, the upper-bound for each entry of $\mathcal{R}[i]$ in search is defined. This paper focuses on following four types of $\mathcal{R}_{top}$:

- $\text{TypeOfBF}_{\text{top}}$ = "Full_BF": This type simply returns $\mathcal{R}_{\text{top}} = [1,1,\dots,1]$ with $\text{P}_{\text{top}} = 1$;
- $\text{TypeOfBF}_{\text{top}}$ = "Step_BF": This type applies a binary search to find the “Step” bounding function’s parameter of $0 < \alpha < 1$ which matches with success probability P_{top}. The cost of this binary search for finding $0 < \alpha < 1$ would be determined in Appendix A as T1 depending on $0 < \text{err}_{\text{layer}} \leq 0.5$ (which is set in line 2 from Algorithm 2 to determine acceptable success probability distance of $0 \leq |p_{succ}(\mathcal{R}_{\text{top}}) - \text{P}_{\text{top}}| < \text{err}_{\text{layer}}$);
- $\text{TypeOfBF}_{\text{top}}$ = "PiecewiseLinear_BF": This type applies a binary search to find a suitable “Piecewise-linear” bounding function’s parameter of $0 < \alpha < 1$ which matches with input success probability P_{top}. The complexity analysis for this type is similar to "Step_BF";
- $\text{TypeOfBF}_{\text{top}}$ = "PrepOpt_BF": This type uses a pre-defined shape of upper-bound for optimal bounding function by authors of this paper from test results:

$$\mathcal{R}[i] = \min\left(1, 10^{-10}\left(\frac{44i}{\beta}\right)^6 - 2\times10^{-8}\left(\frac{44i}{\beta}\right)^5 + 10^{-6}\left(\frac{44i}{\beta}\right)^4 - 2\times10^{-5}\left(\frac{44i}{\beta}\right)^3 + 5\times10^{-4}\left(\frac{44i}{\beta}\right)^2 + 5.8\times10^{-3}\frac{44i}{\beta} + 0.3113581\right), \quad (9)$$

where $1 \leq i \leq \beta$.Suitable form of modifications is defined to re-generate this shape for binary search. The complexity analysis for this type is similar to "Step_BF";

2) Function of $Generate\_\mathcal{R}_{low}$: The lower-bound for each entry $\mathcal{R}[i]$ in search of $\mathcal{R}_{opt}$ is determined by $Generate\_\mathcal{R}_{low}$. For each index $1 \leq i < \beta$, decreasing $\mathcal{R}_{top}[i]$ as $\mathcal{R}'_{top}[i] \leftarrow \mathcal{R}_{low}[i]$, decreases $p_{succ}(\mathcal{R}_{top})$ as $p_{succ}(\mathcal{R}'_{top}) = P_{top} - \frac{P_{top}-P_{low}}{\beta-1}$. By this idea, probability distance between $\mathcal{R}_{low}$ and $\mathcal{R}_{top}$ is divided by $\beta - 1$ columns in range of $\mathcal{R}_{low}[i]$ to $\mathcal{R}_{top}[i]$ for $1 \leq i < \beta$ with success probability length of $column_{width} = \frac{P_{top}-P_{low}}{\beta-1}$. The time complexity of generating $\mathcal{R}_{low}$ is determined in Appendix B.

3) Function of $Generate\_\varepsilon_{array}$: This function tries to divide each of these $\beta - 1$ columns into equal success probability length $step_{length} = \frac{P_{top}-P_{low}}{(\beta-1)\times Layer_{Count}}$ by generating steps of $\boldsymbol{\varepsilon}[l][i]$ as l-th step for decreasing $\mathcal{R}_{top}[i]$ into $\mathcal{R}'_{top}[i] = \mathcal{R}_{top}[i] - \boldsymbol{\varepsilon}[l][i]$ (note that, dimension of matrix of $\boldsymbol{\varepsilon}$ is $Layer_{Count} \times \beta$). This is not necessary to generate all $l \times i$ values of $\boldsymbol{\varepsilon}[l][i]$ at once, this means that at the time each $\boldsymbol{\varepsilon}[l][i]$ is needed, just $\beta - 1$ values of $\boldsymbol{\varepsilon}[l][1]$, ..., $\boldsymbol{\varepsilon}[l][\beta - 1]$ are generated (from top layer to down layer):

- Generating $\boldsymbol{\varepsilon}[1][1], \dots, \boldsymbol{\varepsilon}[1][\beta - 1]$;
- $\vdots$
- Generating $\boldsymbol{\varepsilon}[\text{Layer}_{\text{Count}}][1], \dots, \boldsymbol{\varepsilon}[\text{Layer}_{\text{Count}}][\beta - 1]$;

By decreasing each entry $\mathcal{R}_{\text{top}}[i]$ by $\boldsymbol{\varepsilon}[l][i]$ where $1 \leq l \leq \text{Layer}_{\text{Count}}$, the success probability of $\mathcal{R}_{\text{top}}$ would be decreased by nearly constant unite of $\text{step}_{\text{length}}$. This paper names the batch of all $\text{Layer}_{\text{Count}}$ number of $\text{step}_{\text{length}}$ for each entry $1 \leq i < \beta$ from $\mathcal{R}_{\text{top}}$ as $\text{column}_{\text{width}}$; The time-complexity of generating each layer of $\boldsymbol{\varepsilon}$ is determined in Appendix C as T3;

Note: After $\text{Layer}_{\text{Count}}$ times of decreasing $\mathcal{R}_{\text{top}}[i]$ by $\text{step}_{\text{length}}$, from top to down, they update to $\mathcal{R}_{\text{low}}[i]$.

Note: The acceptable error gaps of $\text{err}_{\text{layer}}$ and $\text{err}_{\text{column}}$ are determined so that total distance of success probability of $\mathcal{R}_{\text{top}}$, $\mathcal{R}_{\text{low}}$ and $\mathcal{R}_{\boldsymbol{\varepsilon}}$ respectively in Algorithm 4, Algorithm 5 and Algorithm 6 would be upper-bounded by $\text{step}_{\text{length}}$;

Note: Let $\mathcal{R}_{\text{low}}[\beta] = \mathcal{R}[\beta] = \mathcal{R}_{opt}[\beta] = \mathcal{R}_{\text{top}}[\beta] = 1$;

4) Function of $Add\_JumpingSteps$: This function adds jumping area with adjustable length and width based on unite of $step_{length}$. Time complexity of this function is determined in Appendix D. Bad jumping parameters may lead to biased search in Algorithm 2 (see Remark 1 in Appendix E)!

5) Function of $Generate\_\mathcal{R}_{opt}\_2$: This function is core of searching in Algorithm 2 and has determinative role in total time-complexity of Algorithm 2. Moreover, this function includes various algorithmic improvements for speedup. Time-complexity of this function is analysed in Appendix E.

```
Algorithm 3 Approximate_$\mathcal{R}_{\text{opt}}$_1
Input: lattice block $\mathcal{L}_\beta$, enum radii $R$, target success prob $P_{\text{final}} \leq 0.5$
1:  $P_{\text{top}} = 2 \times P_{\text{final}}$; $\text{index}_{\text{start}} = \frac{\beta}{2} + 1$;
2:  $\text{Layer}_{\text{Count}} = \beta$; $\text{jump}_{\text{width}} = 1$; $\text{jump}_{\text{len}} = 1$;
3:  $P_{\text{low}} = \frac{P_{\text{top}}}{\text{Layer}_{\text{Count}}+1}$; $\text{err}_{\text{layer}} = \frac{P_{\text{low}}}{\beta}$; $\text{err}_{\text{column}} = \frac{P_{\text{low}}}{\text{Layer}_{\text{Count}}+1}$;
4:  $\mathcal{R}_{\text{top}} \leftarrow \text{Generate\_}\mathcal{R}_{\text{type}}(\beta, P_{\text{top}}, \text{"Step\_BF"}, \text{err}_{\text{layer}})$;
5:  $\mathcal{R}_{\text{low}} \leftarrow \text{Generate\_}\mathcal{R}_{\text{low}}(\beta, P_{\text{low}}, P_{\text{top}}, \mathcal{R}_{\text{top}}, \text{err}_{\text{column}})$;
6:  $\ell_{\text{next}} = 1$; //next layer of steps to decrease entries of $\mathcal{R}$
7:  $\mathcal{R}_{\boldsymbol{\varepsilon}} \leftarrow \mathcal{R}_{\text{top}}$; //$\mathcal{R}_{\boldsymbol{\varepsilon}}[i] \coloneqq \sum_{l=1}^{\ell_{\text{next}}-1}(\mathcal{R}_{\text{top}}[i] - \boldsymbol{\varepsilon}[l][i])$
8:  $\boldsymbol{\varepsilon}[\ell_{\text{next}}][1 \dots \beta] \leftarrow \text{Generate\_}\varepsilon_{\text{array}}(\beta, P_{\text{low}}, \mathcal{R}_{\text{low}}, P_{\text{top}}, \mathcal{R}_{\text{top}}, \mathcal{R}_{\boldsymbol{\varepsilon}}, \text{err}_{\text{layer}}, \ell_{\text{next}})$;
9:  $\ell_{\text{next}}++$;//next layer of steps is set for next call.
10: for $\left(t = 1; t \leq \frac{\beta}{2}; t++\right)\{\mathcal{R}_{\text{initial}}[t] \leftarrow \mathcal{R}_{\text{low}}[t];\}$
11: for $\left(t = \frac{\beta}{2} + 1; t \leq \beta; t++\right)\{\mathcal{R}_{\text{initial}}[t] \leftarrow \mathcal{R}_{\text{top}}[t];\}$
12: Add_JumpingSteps($\beta, \mathcal{R}_{\text{initial}}, \text{jump}_{\text{len}}, \text{jump}_{\text{width}}, \text{index}_{\text{start}}\ \boldsymbol{\varepsilon}, P_{\text{low}}, \mathcal{R}_{\text{low}}, P_{\text{top}}, \mathcal{R}_{\boldsymbol{\varepsilon}}, \mathcal{R}_{\text{top}}, \text{err}_{\text{layer}}, \ell_{\text{next}}$);
13: Add_JumpingSteps($\beta, \mathcal{R}_{\text{initial}}, \text{jump}_{\text{len}}, \text{jump}_{\text{width}}, \text{index}_{\text{start}}\ \boldsymbol{\varepsilon}, P_{\text{low}}, \mathcal{R}_{\text{low}}, P_{\text{top}}, \mathcal{R}_{\boldsymbol{\varepsilon}}, \mathcal{R}_{\text{top}}, \text{err}_{\text{layer}}, \ell_{\text{next}}$);
14: $\mathcal{R}_{\text{A}} \leftarrow \text{Generate\_}\mathcal{R}_{\text{opt}}\text{\_4}(\beta, P_{\text{final}}, \mathcal{R}_{\text{initial}}, \mathcal{R}_{\text{low}})$;
15: $\mathcal{R}_{\text{B}} \leftarrow \text{Generate\_}\mathcal{R}_{\text{type}}(\beta, P_{\text{final}}, \text{"PiecewiseLinear\_BF"}, \text{err}_{\text{layer}})$;
16: if $\left(N(\mathcal{L}_\beta, \mathcal{R}_{\text{A}}, R) \leq N(\mathcal{L}_\beta, \mathcal{R}_{\text{B}}, R)\right)$ $\mathcal{R}_{opt} = \mathcal{R}_{\text{A}}$; else $\mathcal{R}_{opt} = \mathcal{R}_{\text{B}}$;
Output: Approximate optimal bounding function $\mathcal{R}_{opt}$.
```

3.3. An Efficient Technique for Approximating Optimal Bounding Function

Algorithm 3 is our proposed method for generating approximate optimal bounding functions. The design considerations of Algorithm 3 are described briefly as follows.

1) *Function of $Generate\_\mathcal{R}_{type}$*: Let $P_{final} \leq 0.5$ for Algorithm 3 and $P_{top} = 2 \times P_{final}$; Also type of target bounding function is forced to be "$Step\_BF$";
2) *Function of $Generate\_\mathcal{R}_{low}$*: Similar in Algorithm 2;
3) *Function of $Generate\_\varepsilon_{array}$*: Similar in Algorithm 2;
4) *Function of $Add\_JumpingSteps$*: Since $index_{start} = \frac{\beta}{2} + 1$, $Layer_{Count} = \beta$, $jump_{width} = 1$ and $jump_{len} = 1$, so this is possible to call $Add\_JumpingSteps$ for just two times. For correct operation of $Add\_JumpingSteps$, this is necessary to initialize $\mathcal{R}_{initial}$ in line 10 and 11;
5) *Function of $Generate\_\mathcal{R}_{opt}\_2$*: Algorithm 3 introduces acceptable time by removing this function from Algorithm 2.
6) *Function of $Generate\_\mathcal{R}_{opt}\_4$*: This function is final phase of bounding function $\mathcal{R}_A$ to reach the success probability of P_{final}, while introducing least total nodes.
7) *Returned optimal bounding function $\mathcal{R}_{opt}$*: By using $Generate\_\mathcal{R}_{type}$, a piecewise-linear bounding function with success probability of P_{final} is generated as $\mathcal{R}_B$, and finally, one of $\mathcal{R}_A$ and $\mathcal{R}_B$ with less total nodes are returned as $\mathcal{R}_{opt}$.

4. Complexity Analysis

This section analyze time/space complexity of Algorithm 2 and Algorithm 3. However as mentioned at the end of Section 2, our analysis of time complexity in this section is not asymptotic analysis. For analyzing time-complexity of our proposed algorithms, two categories of parameter-set are considered as follows:

- ***Parameter-set of Type 1***: Non-extreme pruning success probability $\mathrm{P_{top}}$ where $\frac{1}{\mathrm{P_{top}}} \approx (\beta) \times \mathcal{C}$; $\mathrm{Layer_{Count}} \approx (\beta) \times \mathcal{C}$; $\mathrm{jump_{width}} \approx (1) \times \mathcal{C}$; $\mathrm{jump_{len}} \approx (1) \times \mathcal{C}$; $\mathrm{step_{count}} \approx (\beta^2/2) \times \mathcal{C}$;
- ***Parameter-set of Type 2***: Extreme pruning success probability $\mathrm{P_{top}}$ where $\frac{1}{\mathrm{P_{top}}} \approx 2^{(\beta)\times\mathcal{C}}$; $\mathrm{Layer_{Count}} \approx (\beta) \times \mathcal{C}$; $\mathrm{jump_{width}} \approx (1) \times \mathcal{C}$; $\mathrm{jump_{len}} \approx (1) \times \mathcal{C}$; $\mathrm{step_{count}} \approx (\beta^2/2) \times \mathcal{C}$;

By Type 1 and Type 2, time-complexity of functions in Algorithm 2 and Algorithm 3 are re-written based on β:

- Algorithm 4 by Parameter-set of Type 1: $\mathrm{T1} \approx \left(\frac{3\beta^3}{8}\log_2 \beta\right) \times \mathcal{C}$, and by Type 2: $\mathrm{T1} \approx \left(\frac{\beta^4}{8}\right) \times \mathcal{C}$;
- Algorithm 5 in Type 1 and Type 2: $\mathrm{T2} \approx \left(\frac{\beta^4}{8}\log_2 \beta\right) \times \mathcal{C}$;
- By assumption of $\mathrm{P_{top}} \approx 2 \times \mathrm{P_{final}}$ and Average-case Assumption 3 (which leads to $\mathrm{Layer_{Count}} \approx (\beta/2) \times \mathcal{C}$), total time-complexity of all calls of Algorithm 6 in Parameter Type 1 and Type 2: $\mathrm{T3_{Total}} \approx \left(\frac{\beta^4}{8}\log_2 \beta\right) \times \mathcal{C}$;
- Algorithm 7 in Parameter-set of Type 1 and Type 2 (in Algorithm 2 with $\mathrm{index_{start}} \approx (1) \times \mathcal{C}$): $\mathrm{T4} \approx \left(\frac{\beta^4 \log_2 \beta}{4}\right) \times \mathcal{C}$;
- Algorithm 7 in Parameter-set of Type 1 and Type 2 (in Algorithm 3 with $\mathrm{index_{start}} = \frac{\beta}{2} + 1$): $\mathrm{T4} \approx \left(\frac{\beta^4 \log_2 \beta}{8}\right) \times \mathcal{C}$;
- Algorithm 8 by ignoring our algorithmic speedups and Average-case Assumption 3 (which leads to $\mathrm{Layer_{Count}} \approx (\beta/2) \times \mathcal{C}$) in Type 1 and Type 2 (in Algorithm 2 and Algorithm 3): $\mathrm{T5_1} \approx \left(\frac{\beta^7 + 16\beta^4 \log_2 \beta}{128}\right) \times \mathcal{C}$;
- Algorithm 8 by using our algorithmic speedups and Average-case Assumption 3 (which leads to $\mathrm{Layer_{Count}} \approx (\beta/2) \times \mathcal{C}$) in Type 1 and Type 2: $\mathrm{T5_2} \approx \left(\frac{\beta^7 + 2.\overline{66}\beta^6 + 1.\overline{66}\beta^5 + 40\beta^4 \log_2 \beta + 80\,\beta^4}{320}\right) \times \mathcal{C}$;
- Algorithm 10 with Average-case Assumption 4 in Type 1 and Type 2 (in Algorithm 3): $\mathrm{T7} \approx (\beta^4/32) \times \mathcal{C}$;

By using these time-complexities, total time-complexity of Algorithm 2 and Algorithm 3 can be determined as follows:

- Algorithm 2 by using our algorithmic speedups in Parameter-set of Type 1 and Type 2:

$$\mathrm{T}_{\mathrm{Generate}\_\mathcal{R}_{\mathrm{opt}\_3}} \approx \mathrm{T1} + \mathrm{T2} + \mathrm{T3_{Total}} + \mathrm{T4} + \mathrm{T5_2} \approx \left(\frac{\beta^7 + 2.\overline{66}\beta^6 + 1.\overline{66}\beta^5 + 200\beta^4 \log_2 \beta + 120\,\beta^4}{320}\right) \times \mathcal{C}. \tag{10}$$

- Algorithm 3 in Parameter-set of Type 1 and Type 2:

$$\mathrm{T}_{\mathrm{Approximate}\_\mathcal{R}_{\mathrm{opt}\_1}} = \mathrm{T1} + \mathrm{T2} + \mathrm{T3_{Total}} + 2\mathrm{T4} + \mathrm{T7} + \mathrm{T1} \approx \left(\frac{240\beta^4 \log_2 \beta + 90\beta^4}{320}\right) \times \mathcal{C}. \tag{11}$$

In other side, the space-complexity of Algorithm 2 and Algorithm 3 can be determined simply just by computing the size of matrix of $\boldsymbol{\varepsilon}[\ell = 1 \ldots \mathrm{Layer_{Count}}][1 \ldots \beta]$ as follows:

$$\mathrm{S}_{\mathrm{Generate}\_\mathcal{R}_{\mathrm{opt}\_3}} \approx \mathrm{S}_{\mathrm{Approximate}\_\mathcal{R}_{\mathrm{opt}\_1}} \approx \left(\mathrm{Layer_{Count}} \times \beta \times \mathrm{sizeof(Real\_number)}\right) \times \mathcal{C}. \tag{12}$$

For example, for $\beta = 300$ and $\mathrm{Layer_{Count}} = 300$, and x64 bit platform (by assuming sizeof(Real_number) = 64 bit), space complexity of Algorithm 1 and Algorithm 2 are $\approx$ 703 KB.

5. Our Results

Our test results in this section try to justify and verify the value of our contributions in this paper. All the tests in this paper are run on the random instances of SVP lattice challenges [11]. Also, all the implementations are compiled with MSVC x64 bit C++. These tests use the following hardware platform: ASUS motherboard series Z97-K, Intel® Core™ i7-4790K processor with the base frequency of 4 GHz, 16 GB RAM; also, the running times are provided only for a single real-core.

5.1. Shape of Optimal Bounding Function Candidates

For better sense on our test results, the shape of discussed bounding functions is shown in this sub-section. All candidates of generating optimal bounding function in this paper include:

- ***Chen_Nguyen_BF***: This type of optimal bounding are generated by Algorithm 5 from Appendix A of [6];
- ***R_BF1***: This type of optimal bounding are generated as $\mathcal{R}_{\mathrm{A}}$ in line 14 from Algorithm 3 in this paper;
- ***R_BF2***: This type of bounding are generated similar to $\mathcal{R}_{\mathrm{A}}$ in line 14 from Algorithm 3, except that instead of Add_JumpingSteps from Algorithm 3, we use a function to make the shape of $\mathcal{R}_{\mathrm{A}}$ similar to Algorithm 5 from [6];
- ***PieceWise_Linear_BF***: See definition in Section 5.3 from [5] (this function is used in line 15 from Algorithm 3);
- ***Step_BF***: See the original definition in Section 5.3 of [5];
- ***BF[from Local_Search1]***: This type of bounding functions are generated by Algorithm 2 for $\mathrm{TypeOfBF}_{\mathrm{top}} =$ "Full_BF" (see Section 3.2).
- ***BF[from Local_Search2]***: This type of bounding functions are generated by Algorithm 2 with $\mathrm{TypeOfBF}_{\mathrm{top}} =$ "Full_BF" together with applying Add_JumpingSteps;
- ***BF[from Local_Search3]***: This type of bounding functions are generated by Algorithm 2 for $\mathrm{TypeOfBF}_{\mathrm{top}} =$ "Step_BF" (see in Section 3.2);
- ***BF[from Local_Search4]***: This type of bounding functions are generated by Algorithm 2 with $\mathrm{TypeOfBF}_{\mathrm{top}} =$ "Step_BF" together with applying Add_JumpingSteps;
- ***BF[from Local_Search5]***: This type of bounding functions are generated by Algorithm 2 for $\mathrm{TypeOfBF}_{\mathrm{top}} =$ "PrepOpt_BF" together with applying Add_JumpingSteps;

Fig.1 shows the shape of these candidates for $p_{succ} = 0.25$. Also Fig.2 and Fig.3 respectively show generated optimal bounding functions for $p_{succ} = 0.25$ by Algorithm 2 [from Local_Search5] and Algorithm 3, including entries lower bounded by $\mathcal{R}_{\mathrm{low}}$ and upper-bounded by $\mathcal{R}_{\mathrm{top}}$.

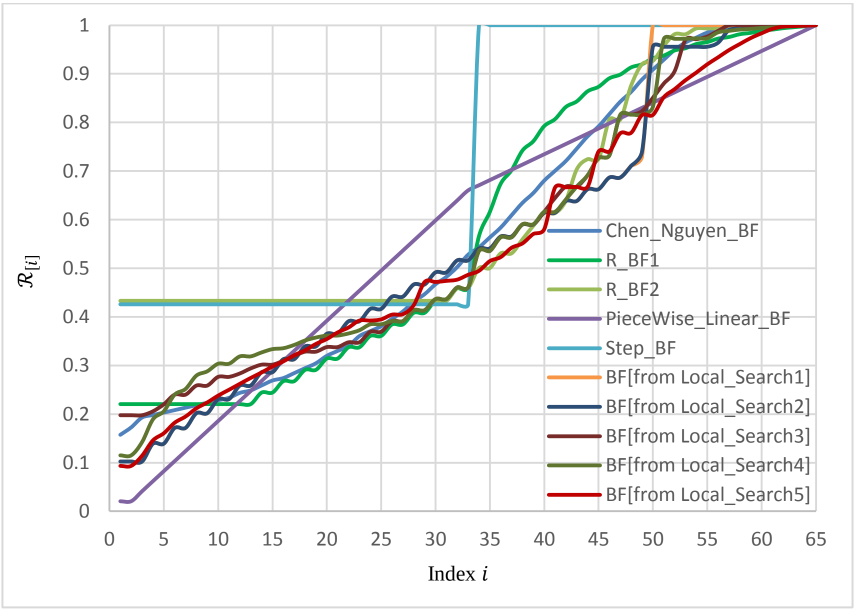

Fig.1. Candidates of optimal bounding function for $p_{succ} = 0.25$

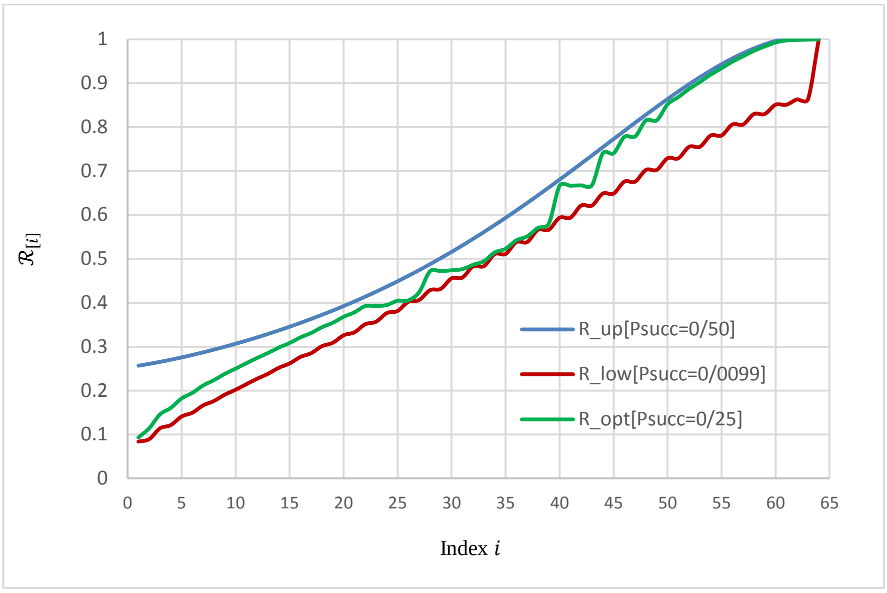

Fig.2. Optimal bounding function by Local_Search5 for $p_{succ} = 0.25$

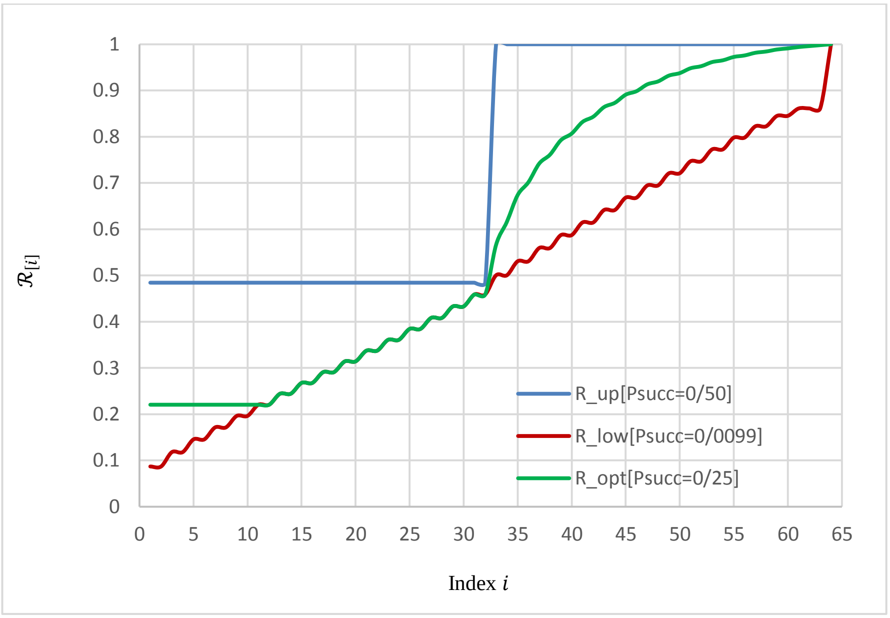

Fig.3. Optimal bounding function by Algorithm 3 for $p_{succ} = 0.25$

5.2. Results for Quality of Generated Bounding Functions

To have better sense on running time of our proposed candidates of generators of optimal bounding function in this paper, all of them are tested to generate an optimal bounding function with success probability $p_{succ} = 0.0001$ over nearly 20 SVP random lattice challenges [11] with block size of $\beta = 100$. Table 1 shows the running time of these generators. As shown in Table 1, the running time of generating Step bounding function and Piecewise-linear bounding function by their original definition in [5] (respectively as “Step_BF” and “PieceWise_Linear_BF”) is negligible, since they are not search algorithm at all. Also, since Algorithm 3 is combined by definition of “PieceWise_Linear_BF” and generator of “R_BF1”, the running time of Algorithm 3 in Table 1 is sum of the running time of these two generators. Moreover, this is clear that, since the generators of “Local_Search1”, …, “Local_Search5” directly search the optimal bounding functions (against our other proposed generators which just generate or approximate these bounding functions), their running times are much more than our other proposed generators.

To have better comparison between the proposed candidates of optimal bounding function (in the previous section), GNR-enumeration is run practically by these bounding functions over some SVP challenges [11] and exact number of nodes in corresponding enumeration tree are counted. The results of this experimental test are shown in Table 2 for block size of 65. As shown in Table 2, our approximate technique in Algorithm 3 introduces acceptable low cost (closed to Chen-Nguyen technique in Algorithm 5 from [6]). Also Algorithm 2 from type of "Local_Search5" in this test introduces better cost than Algorithm 3 for $p_{succ} \leq 0.05$.

As mentioned in Section 4, time complexity of Algorithm 8 by ignoring our algorithmic speedups and the one by using our algorithmic speedups are nearly differed by some factor of Speedup $\approx O(1)$ which is non-negligible. Table 2 compares Algorithm 2 by using our algorithmic speedups and ignoring these algorithmic speedups for the generators of "Local_Search1", …, "Local_Search5". As shown in our partial test results for dimension of $\beta = 65$ in Table 2, running time of Algorithm 8 by ignoring these speedups are more than running time of Algorithm 8 by considering these speedups, nearly by some factors of $2.45 <$ Speedup < 4.26 which is non-negligible while it can be considered as Speedup $\approx O(1)$ in asymptotic analysis (note that our analysis of time complexity in Section 4 is not asymptotic analysis).

Table 1. Running-Time (in seconds) of our proposed candidates of generators of optimal bounding function over SVP challenges with $\beta \approx 100$

Psucc / Bound. Func.	0/0001
BF1	32
PieceWise_BF	0.077
Algorithm 3	32.077
BF2	25
Step_BF	0.038
Local_Search1	64625
Local_Search2	38027
Local_Search3	19130
Local_Search4	9342
Local_Search5	4871

Table 2. log_2 of number of tree nodes (log_2 of cost) in GNR-enumeration pruned by some candidates of optimal bounding function with $\beta \approx 65$

Psucc / Bound. Func.	0/0001	0/001	0/01	0/02	0/03	0/04	0/05	0/25	0/50	0/95
BF1 as $\mathcal{R}_A$	9/76	10	11	11/5	11/7	11/9	12/4	14/7	17	-
PieceWise_BF as $\mathcal{R}_B$	7/95	8/82	10/3	11/2	11/8	12/3	12/8	17/4	21/4	-
Algorithm 3 as $\min(\mathcal{R}_A, \mathcal{R}_B)$	7/95	8/82	10/3	11/2	11/7	11/9	12/4	14/7	17	
Chen_Nguyen	-	-	10	-	-	-	11/8	15/3	-	24
BF2	11/4	12/8	15/4	14/9	16/8	15/7	17/4	21/4	25/8	-
Step_BF	12/6	13/9	16	16/9	17/4	17/7	18/1	21/2	23/2	27/5
Local_Search1	-	-	-	-	-	-	-	16/4	17	29/6
Local_Search2	-	-	-	-	-	-	-	16/4	19/7	25/3
Local_Search3	9/13	10/5	12/3	12/4	13/7	13/8	14/3	16/5	18	25/8
Local_Search4	9/53	10/6	12/7	13/3	13/5	13/8	14/4	17/3	19/1	25
Local_Search5	7/75	8/64	10	10/6	11/3	11/7	12/1	16/3	18/3	24/7

Table 3. Running-Time (in seconds) of Algorithm 8 $Generate\_\mathcal{R}_{opt\_}2$ for different configurations of Algorithm 2 with $\beta \approx 65$

Psucc / Bounding Function	0/0001	0/001	0/01	0/02	0/03	0/04	0/05	0/25	0/50	0/95
Local_Search1	-	-	-	-	-	-	-	4417	2759	1418
Local_Search1 (no improve)	-	-	-	-	-	-	-	12881	6942	6040
Local_Search2	-	-	-	-	-	-	-	1513	1519	1144
Local_Search2 (no improve)	-	-	-	-	-	-	-	6023	6072	3834
Local_Search3	1507	1524	1516	1430	1463	1550	1513	1450	1517	1198
Local_Search3 (no improve)	4558	4060	4402	4484	4515	4580	4512	4325	4069	3279
Local_Search4	630	574	543	551	432	622	591	534	625	241
Local_Search4 (no improve)	1806	1873	1925	1588	1596	1595	1513	1504	1597	745
Local_Search5	213	238	273	237	241	250	287	385	840	324
Local_Search5 (no improve)	743	832	856	967	955	966	1084	1393	2060	1092

6. Summary, Conclusion, and Future Works

GNR-enumeration as a claimant type of lattice enumeration is defined based on the pruning by bounding function vector. Using an optimal bounding function to prune the most number of nodes in enumeration tree is one of the main task for running GNR-enumeration on practical block sizes. Generation of optimal bounding function for big block sizes needs to analyze the running time in order that these bounding functions would be generated in less running time. Unfortunately, the running time of the original proposed algorithm of searching optimal bounding function by the work of Chen-Nguyen in [6] is not analyzed at all. In fact, this paper proposes more efficient generators of optimal bounding function to minimizing the cost (running time) of GNR-enumeration, while the time/space complexity of our generators are analyzed accurately. To the best of our knowledge, analyzing and improving the original generator of optimal bounding function in [6] are not discussed in former studies.

In this paper, a global search (Algorithm 1) is introduced which claims to generate the optimal bounding function by a greedy strategy, but unsuitable steps in this algorithm (Algorithm 1) make it's running time intolerable for practical purpose. Moreover, the time-complexity of this algorithm cannot be analyzed accurately. Re-defining the searching steps based on success probability of bounding functions (instead of differences of entries in bounding functions in Algorithm 1) makes it possible to analyze the time-complexity of searching optimal bounding function. By using the basic idea behind our global search (Algorithm 1) together with using the searching steps based on success probability, this paper introduce a new algorithm (Algorithm 2) in search of optimal bounding function. For some proposed input parameter-set in Section 4, the space (memory)-complexity and time-complexity of our search algorithm (Algorithm 2) respectively are $\left(\beta^2 \text{sizeof(Real\_number)}\right) \times \mathcal{C}$ and $\left(\frac{\beta^7+2.\overline{66}\beta^6+1.\overline{66}\beta^5+200\beta^4\log_2\beta+120\,\beta^4}{320}\right) \times \mathcal{C}$. Some of the main superiorities of our new algorithm include:

- Possiblity of jumping from local optimums;
- Simple heuristics to guide search to global optimum;
- Useful input parameters which can be tuned to make a trade-off between optimality of output and running time.
- The useful algorithmic techniques for better speedup;

Furthermore, by using the building blocks in this new search algorithm (Algorithm 2), another new algorithm (Algorithm 3) with lower-time complexity is designed to approximate the optimal bounding function with acceptable quality. For some proposed input parameter-set in Section 4, the space (memory)-complexity and time-complexity of our approximate method (Algorithm 3) respectively are $\left(\frac{240\beta^4\log_2\beta+90\beta^4}{320}\right) \times \mathcal{C}$, and $(\text{sizeof(Real\_number)} \times \beta^2) \times \mathcal{C}$. Our test results in this paper justify and verify the acceptable running time of these two algorithms of search (Algorithm 2) and approximate (Algorithm 3), also the acceptable cost of enumerations which use these output bounding functions.

However this paper introduces some suitable generators for optimal bounding function in GNR-enumeration, more efficient and more exact generators of optimal bounding function are expected to be studied and proposed in future researches. Also this is possible to use formal methods by some theorem prover such as Isabelle/HOL in proving Conjecture 1 by using the basic ideas, structures and propositions which are proposed in [12].

Appendices

Appendix A. Generating well-defined bounding function

Algorithm 4 efficiently generates a bounding function from well-defined types (e.g., Step bounding function).

```
Algorithm 4 Generate_$\mathcal{R}_{\text{type}}$
Input: block size $\beta$, target success probablity $\mathrm{P_T}$, TypeOfBF, $\mathrm{err_{layer}}$
1:  $\mathcal{R} \leftarrow$ InitialBoudingFunc($\beta$, TypeOfBF);
2:  $\mathrm{P} \leftarrow p_{succ}(\mathcal{R})$;
3:  step $\leftarrow$ InitialStep($\beta$, TypeOfBF);
4:  while$(|\mathrm{P_T} - \mathrm{P}| > \mathrm{err_{layer}})${
5:      if$(\mathrm{P_T} - \mathrm{P} < -\mathrm{err_{layer}})${MoveDownBF($\mathcal{R}$, step); }
6:      else if$(\mathrm{P_T} - \mathrm{P} > \mathrm{err_{layer}})${MoveUpBF($\mathcal{R}$, step); }
7:      step $= \frac{\text{step}}{2}$; $\mathrm{P} \leftarrow p_{succ}(\mathcal{R})$;
8:  }//end while
Output: requested bounding function $\mathcal{R}$.
```

Time-complexity of Algorithm 4 is determined as follows:

$$1 \approx \left(\frac{\beta^3}{8}\log_2\left(\frac{1}{\mathrm{err_{layer}}}\right)\right) \times \mathcal{C} \approx \left(\frac{\beta^3}{8}\log_2\left(\frac{\beta(\mathrm{Layer_{Count}}+1)}{\mathrm{P_{top}}}\right)\right) \times \mathcal{C} \Rightarrow$$
$$\mathrm{T1} \approx \left(\frac{\beta^3}{8}\log_2\left(\frac{\beta\ \mathrm{Layer_{Count}}}{\mathrm{P_{top}}}\right)\right) \times \mathcal{C}. \tag{13}$$

Appendix B. Generating lower-bound bounding function

Algorithm 5 efficiently generates the bounding function $\mathcal{R}_{\mathrm{low}}$ as the lower-bounds for entries of searched bounding function in Algorithm 1, Algorithm 2 and Algorithm 3.

```
Algorithm 5 Generate_$\mathcal{R}_{\mathrm{low}}$
Input: $block\ size\ \beta, probablity\ \mathrm{P_{low}}, \mathrm{P_{top}}, \mathcal{R}_{\mathrm{top}}, \mathrm{err_{column}}$
1:    $\mathcal{R} \leftarrow \mathcal{R}_{\mathrm{top}}$; $\mathrm{P} \leftarrow \mathrm{P_{top}}$; $\mathrm{column_{width}} \leftarrow \frac{\mathrm{P_{top}}-\mathrm{P_{low}}}{\beta-1}$;
2:    for$(i = 1; i \le \beta - 1; i++)$\{$//Note: \mathcal{R}[\beta] = 1$
3:        step $\leftarrow \frac{\mathcal{R}_{\mathrm{top}}[i]}{2}$;
4:        while$(|\mathrm{P_{top}} - \mathrm{P} - \mathrm{column_{width}} \times i| > \mathrm{err_{column}})$\{
5:            if$(\mathrm{P_{top}} - \mathrm{P} - \mathrm{column_{width}} \times i < -\mathrm{err_{column}})$\{$\mathcal{R}[i]$-= step;\}
6:            else if$(\mathrm{P_{top}} - \mathrm{P} - \mathrm{column_{width}} \times i > \mathrm{err_{column}})$\{$\mathcal{R}[i]$+= step;\}
7:            step $\leftarrow \frac{\mathrm{step}}{2}$; P $\leftarrow p_{succ}(\mathcal{R})$;
8:        \}//end while
9:    \}//end for
Output: bounding function $\mathcal{R}_{\mathrm{low}} = \mathcal{R}$.
```

Time-complexity of Algorithm 5 is determined as follows:

$$\mathrm{T2} \approx \left((\beta - 1) \times \frac{\beta^3}{8}\log_2\left(\frac{\mathrm{column_{width}}}{\mathrm{err_{column}}}\right)\right) \times \mathcal{C} \approx \left(\frac{\beta^4}{8}\log_2\left(\frac{\mathrm{P_{top}}-\mathrm{P_{low}}}{\beta-1} \times \frac{(\mathrm{Layer_{Count}}+1)^2}{\mathrm{P_{top}}}\right)\right) \times \mathcal{C} \Rightarrow$$
$$\mathrm{T2} \approx \left(\frac{\beta^4}{4}\log_2 \mathrm{Layer_{Count}} - \frac{\beta^4}{8}\log_2 \beta\right) \times \mathcal{C}. \tag{14}$$

Appendix C. Efficient Generator for matrix of $\boldsymbol{\varepsilon}$

Algorithm 6 efficiently generates the matrix of $\boldsymbol{\varepsilon}$ as dynamic steps in search of optimal bounding function.

```
Algorithm 6 Generate_$\varepsilon_{\mathrm{array}}$
Input: $\beta, \mathrm{P_{low}}, \mathcal{R}_{\mathrm{low}}, \mathrm{P_{top}}, \mathcal{R}_{\mathrm{top}}, \mathcal{R}_{\boldsymbol{\varepsilon}}, \mathrm{err_{layer}}$, layer index of $\ell_{\mathrm{next}}$
1:    $\mathcal{R} \leftarrow \mathcal{R}_{\mathrm{top}}$; $\mathrm{P} \leftarrow \mathrm{P_{top}}$; $\mathrm{P_T} = \mathrm{P_{top}} - \ell_{\mathrm{next}} \times \mathrm{P_{low}}$;
2:    $\mathrm{step_{vec}} \leftarrow \frac{\mathcal{R}_{\mathrm{top}}-\mathcal{R}_{\mathrm{low}}}{2}$;
3:    while$(|\mathrm{P_T} - \mathrm{P}| > \mathrm{err_{layer}})$\{
4:        if$(\mathrm{P_T} - \mathrm{P} < -\mathrm{err_{layer}})$\{MoveDownBF$(\mathcal{R}, \mathrm{step_{vec}})$;\}
5:        else if$(\mathrm{P_T} - \mathrm{P} > \mathrm{err_{layer}})$\{MoveUpBF$(\mathcal{R}, \mathrm{step_{vec}})$;\}
6:        $\mathrm{step_{vec}} = \frac{\mathrm{step_{vec}}}{2}$; P $\leftarrow p_{succ}(\mathcal{R})$;
7:    \}//end while
8:    for$(i = 1; i \le \beta; i++)$ \{$\varepsilon_{\mathrm{current}}[i] \leftarrow \mathcal{R}_{\boldsymbol{\varepsilon}}[i] - \mathcal{R}[i]$;\}
9:    $\mathcal{R}_{\boldsymbol{\varepsilon}} \leftarrow \mathcal{R}$; $//\mathcal{R}_{\boldsymbol{\varepsilon}}$ is set for next call.
Output: last step vectors of $\varepsilon_{\mathrm{current}}$ $as$ $\boldsymbol{\varepsilon}[\ell_{\mathrm{next}}][1 \dots \beta]$.
```

Time-complexity of Algorithm 6 is determined as follows:

$$\mathrm{T3} \approx \left(\frac{\beta^3}{8}\log_2\left(\frac{\mathrm{P_{top}}-\mathrm{P_{low}}}{\mathrm{err_{layer}}}\right)\right) \times \mathcal{C} \approx \left(\frac{\beta^3}{8}\log_2\left(\frac{(\mathrm{P_{top}}-\mathrm{P_{low}})\times\beta}{\mathrm{P_{low}}}\right)\right) \times \mathcal{C} \Rightarrow$$
$$\mathrm{T3} \approx \left(\frac{\beta^3}{8}\log_2(\beta \times \mathrm{Layer_{Count}})\right) \times \mathcal{C}. \tag{15}$$

Assumption 1 [Worst-case]. Time-complexity for all calls of Algorithm 6 in Algorithm 2 is $\mathrm{T3_{Total}} \approx (\mathrm{Layer_{Count}} \times \mathrm{T3}) \times \mathcal{C}$.

Appendix D. Adding jumping areas in search space

Algorithm 7 adds a suitable form of jumping on initial bounding function before search in Algorithm 2. Since the function of Generate_$\varepsilon_{\text{array}}$ is called one time in each *for3* (in line 3) from Algorithm 7, the time complexity of this algorithm (Algorithm 7) is determined as follows:

$$4 \approx \sum_{i=1}^{\text{jump}_{\text{width}}} \sum_{j=1}^{\frac{\beta-\text{index}_{\text{start}}}{\text{jump}_{\text{len}}}} \left(\left(\frac{\beta^3}{8} \log_2(\beta \times \text{Layer}_{\text{Count}}) \right) \times \mathcal{C} + \sum_{k=\text{index}_{\text{start}}}^{\beta - j \times \text{jump}_{\text{len}}} (1) \times \mathcal{C} \right) \Rightarrow$$

$$\text{T4} \approx \left(\frac{\text{jump}_{\text{width}} \times \beta^3 \times (\beta - \text{index}_{\text{start}}) \times \log_2(\beta \times \text{Layer}_{\text{Count}})}{8\ \text{jump}_{\text{len}}} \right) \times \mathcal{C}. \tag{16}$$

Assumption 2 [Worst-case]. The maximum time complexity of Algorithm 7 can be observed by letting the parameters of $\text{jump}_{\text{width}} = \frac{\text{Layer}_{\text{Count}}+1}{\beta}$, $\text{jump}_{\text{len}} = 1$;

Algorithm 7 Add_JumpingSteps

```
Input: block size $\beta$, $\mathcal{R}$, $\text{jump}_{\text{len}}$, $\text{jump}_{\text{width}}$, $\text{index}_{\text{start}}$, $\boldsymbol{\varepsilon}$, $\text{P}_{\text{low}}$, $\mathcal{R}_{\text{low}}$, $\text{P}_{\text{top}}$, $\mathcal{R}_{\boldsymbol{\varepsilon}}$, $\mathcal{R}_{\text{top}}$, $\text{err}_{\text{layer}}$, $\ell_{\text{next}}$.
 1:  for($i = 1; i \le \text{jump}_{\text{width}}; i++$){//for1
 2:    for($j = \text{jump}_{\text{len}}; j \le \beta - \text{index}_{\text{start}}; j += \text{jump}_{\text{len}}$){//for2
 3:      for($k = \text{index}_{\text{start}}; k \le \beta - j; k++$){//for3
 4:        if($0 < \mathcal{R}_{k-1}^2 \le \mathcal{R}_k^2 - \boldsymbol{\varepsilon}[\varepsilon_{\text{layer}}[k]][k]$){//if1
 5:          $\mathcal{R}_k^2 = \mathcal{R}_k^2 - \boldsymbol{\varepsilon}[\varepsilon_{\text{layer}}[k]][k]$;
 6:          $\varepsilon_{\text{layer}}[k]++$;
 7:          if($\varepsilon_{\text{layer}}[i] = \ell_{\text{next}}$){//if2
 8:            $\boldsymbol{\varepsilon}[\ell_{\text{next}}][1 \ldots \beta] \leftarrow$ Generate_$\varepsilon_{\text{array}}(\beta, \text{P}_{\text{low}}, \mathcal{R}_{\text{low}}, \text{P}_{\text{top}}, \mathcal{R}_{\boldsymbol{\varepsilon}}, \mathcal{R}_{\text{top}}, \text{err}_{\text{layer}}, \ell_{\text{next}})$;
 9:            $\ell_{\text{next}}++$;}/* end if1 */}/* end if2 */}//end for3
10:  }/* end for2 */}//end for1
Output: bounding function $\mathcal{R}$ with jumping steps
```

To have a better sense on application of this function, see resulted bounding function for $\beta = 65$ after applying Add_JumpingSteps in Fig.4:

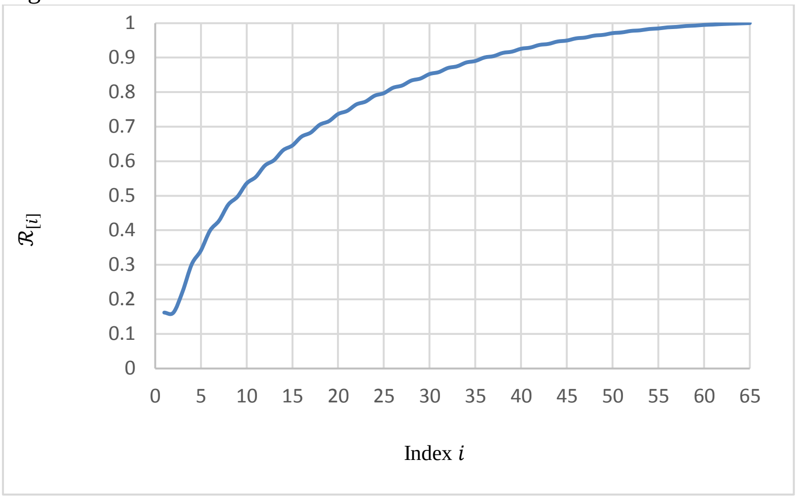

Fig.4. Resulted bounding function after Add_JumpingSteps

Appendix E. Efficient search of $\mathcal{R}_{\text{opt}}$ in Algorithm 2

Algorithm 8 is the core of search in Algorithm 2, and includes useful algorithmic improvements for speedup. Time complexity of Algorithm 8 is determined as follows. By applying Algorithm 7 (Add_JumpingSteps), the probability of $\text{P}_{\text{initial}} = p_{succ}(\mathcal{R}_{\text{initial}})$ would be estimated as follows:

$$\text{P}_{\text{initial}} \approx \text{P}_{\text{top}} - \text{step}_{\text{length}} \times \sum_{i=1}^{\text{jump}_{\text{width}}} \sum_{j=1}^{\frac{\beta-\text{index}_{\text{start}}}{\text{jump}_{\text{len}}}} \sum_{k=\text{index}_{\text{start}}}^{\beta - j \times \text{jump}_{\text{len}}} (1) \times \mathcal{C}.$$

Let $\text{P}_{\text{final}} \le 0.50$ and consider $\text{P}_{\text{top}} = \min(0.9999, 2 \times \text{P}_{\text{final}})$ in line 1 from Algorithm 2 (Generate_$\mathcal{R}_{\text{opt}}$_3):

$$\mathrm{P_{final}} = \frac{\beta-1}{2} \times (\mathrm{Layer_{Count}} + 1) \times \mathrm{step_{length}} \Rightarrow$$

$$\mathrm{P_{initial}} \approx \mathrm{step_{length}} \left((\beta - 1) \times (\mathrm{Layer_{Count}} + 1) - \frac{1}{2}\mathrm{jump_{width}} \times \left(\frac{\beta - \mathrm{index_{start}}}{\mathrm{jump_{len}}}\right) (\beta - \mathrm{index_{start}} - \mathrm{jump_{len}} + 2) \right) \Rightarrow$$

$$\mathrm{step_{count}} = \frac{\mathrm{P_{initial}} - \mathrm{P_{final}}}{\mathrm{step_{length}}} = \left(\frac{1}{2}(\beta - 1) \times (\mathrm{Layer_{Count}} + 1) - \frac{1}{2}\mathrm{jump_{width}} \times \left(\frac{\beta - \mathrm{index_{start}}}{\mathrm{jump_{len}}}\right) (\beta - \mathrm{index_{start}} - \mathrm{jump_{len}} + 2) \right).$$

Remark 1. By using Worst-case Assumption 2, and $\mathrm{index_{start}} = 1$, time complexity of Algorithm 8 Generate_$\mathcal{R}_{\mathrm{opt}}$_2 would be zero (i.e., Algorithm 8 doesn't operate), since $\mathrm{step_{count}}$ by this parameter-set becomes zero.

At first, the time complexity of Algorithm 8 by ignoring our algorithmic speedups is determined as follows:

$$\mathrm{T5_1} \approx \mathrm{T3_{Total}} + \sum_{\mathrm{step}=1}^{\mathrm{step_{count}}} \left(\sum_{i=1}^{\beta} \left((i) \times \mathcal{C} + \sum_{t=1}^{\beta/2} \left(\frac{(2t)^3}{8}\right) \times \mathcal{C} \right) + \sum_{i=1}^{\beta} \sum_{t=1}^{\beta} (1) \times \mathcal{C} \right) \Rightarrow$$

$$\mathrm{T5_1} \approx (\mathrm{Layer_{Count}} \times \mathrm{T3}) \times \mathcal{C} + \left(\mathrm{step_{count}} \times \frac{\beta^5}{64} \right) \times \mathcal{C}. \tag{17}$$

At next, the time complexity of Algorithm 8 by using our algorithmic speedups is determined as follows:

$$\mathrm{T5_2} \approx \mathrm{T3_{Total}} + \sum_{\mathrm{step}=1}^{\mathrm{step_{count}}} \left(\sum_{i=1}^{\beta} \left((i) \times \mathcal{C} + \sum_{t=i/2}^{\beta/2} \left(\frac{(2t)^2 i}{8}\right) \times \mathcal{C} \right) + \sum_{i=1}^{\beta} \sum_{t=i+1}^{\beta} (1) \times \mathcal{C} + \mathrm{Avegrage}_{\mathrm{S_{id}}=1}^{\beta-1} \left[\sum_{t=\lceil \frac{\mathrm{S_{id}}}{2} \rceil}^{\frac{\beta}{2}} \left(\frac{(2t)^2 \mathrm{S_{id}}}{8}\right) \times \mathcal{C} \right] \right) \Rightarrow$$

Algorithm 8 Generate_$\mathcal{R}_{\mathrm{opt}}$_2

```
Input: lattice block $\mathcal{L}_\beta$, enum radii $R$, success prob $\mathrm{P_{final}}$, error array $\boldsymbol{\varepsilon}$, next layer of steps as $\ell_{\mathrm{next}}, \mathcal{R}_{[0,1,\dots,\beta]}$,   /* initial bounding function which is used in search, while $\mathcal{R}[0] = 0^+$ is set only for correctness */, $\mathrm{Layer_{Count}}$ /*max number of steps*/.
1:  Generate_structure([C, XY, $i$]); //C, XY and $i$ is defined in Algorithm 9
2:  $N_{\mathrm{Min}[0,1,2,\dots,\beta]} = [0,0,\dots,0]$;  $\mathrm{more_{Improve}} = true$; //$N_{\mathrm{Min}}[0]$ is always set to 0 for correctness of algorithm
3:  $\varepsilon_{\mathrm{layer}\,[1,2\dots,\mathrm{Layer_{Count}}]} = [1,1,1,\dots,1]$;
4:  for($t = 1; t \le \beta; t++$) {$\mathrm{gh}_t = \mathrm{GH}(\mathcal{L}_t)$; }
5:  for($t = 1; t \le \beta;$ ){//for1
6:     $\mathrm{Pr}[t] = \mathrm{Pr}[t+1] = p_{succ}2(\mathcal{L}_{t+1}, \mathcal{R}_{[1\dots t+1]}, [\mathrm{C}, -\mathrm{w}, i])$;
7:     $V[t] = \left(\frac{R\,\mathcal{R}_t}{\mathrm{gh}_t}\right)^t$; $N_{\mathrm{Min}}[t] = N_{\mathrm{Min}}[t-1] + \frac{1}{2} V[t]\,\mathrm{Pr}[t]$;
8:     $t++$;
9:     $V[t] = \left(\frac{R\,\mathcal{R}_t}{\mathrm{gh}_t}\right)^t$; $N_{\mathrm{Min}}[t] = N_{\mathrm{Min}}[t-1] + \frac{1}{2} V[t]\,\mathrm{Pr}[t]$;
10:    $t++$}//end for1
11: $P_{\mathrm{last}} = \mathrm{Pr}[\beta]$; $N_{\mathrm{last}} = N_{\mathrm{Min}}[\beta]$;
12: while($\mathrm{more_{Improve}} = true$){//while1
13:    $\mathrm{more_{Improve}} = false$;
14:    for($i = 1; i \le \beta;$ ){//for2
15:       if($\mathcal{R}_i^2 - \boldsymbol{\varepsilon}[\varepsilon_{\mathrm{layer}}[i]][i] \le 0$) {$i++$; continue in line 14; }
16:       $N[i][i-1] = N_{\mathrm{Min}}[i-1]$; $\mathcal{R}_i^2 = \mathcal{R}_i^2 - \boldsymbol{\varepsilon}[\varepsilon_{\mathrm{layer}}[i]][i]$;
17:       $V_0 = \left(\frac{R\,\mathcal{R}_i}{\mathrm{gh}_i}\right)^i$; $t = i$;
18:       if($2 \times \lceil i/2 \rceil = i$){//i.e., whether $i$ is even or not
19:          $N[i][t] = N[i][t-1] + \frac{1}{2} V_0\,\mathrm{Pr}[t]; t++; V_0 = V[t]$; }
20:       for(; $t < \beta$; ){//for3
21:          if($2 \times \lceil i/2 \rceil = i + 1$){//i.e., whether $i$ is odd or not
22:             $\mathrm{Pr}[t] = \mathrm{Pr}[t+1] = p_{succ}2(\mathcal{L}_{t+1}, \mathcal{R}_{[1\dots t+1]}, [\mathrm{C}, \mathrm{r}-, i+1])$; }//end if
23:          $N[i][t] = N[i][t-1] + \frac{1}{2} V_0\,\mathrm{Pr}[t]; t++; V_0 = V[t]$;
24:          $N[i][t] = N[i][t-1] + \frac{1}{2} V_0\,\mathrm{Pr}[t]; t++; V_0 = V[t]$;
25:       }//end for3
26:       $P[i] = \mathrm{Pr}[\beta]; \mathcal{R}_i^2 = \mathcal{R}_i^2 + \boldsymbol{\varepsilon}[\varepsilon_{\mathrm{layer}}[i]][i]; i++$; }//end for2
27:    for($i = 1; i \le \beta; i++$){//for4
28:       $\mathrm{IsMin} = true$;
29:       if($P_{\mathrm{last}} = P[i]$) {NoOperation;/* $\mathrm{IsMin} = true$ */}
30:       else if($\mathcal{R}_{i-1}^2 > \mathcal{R}_i^2 - \boldsymbol{\varepsilon}[\varepsilon_{\mathrm{layer}}[i]][i]$ or $\mathcal{R}_i^2 - \boldsymbol{\varepsilon}[\varepsilon_{\mathrm{layer}}[i]][i] \le 0$ or $P[i] < \mathrm{P_{final}}$) {$\mathrm{IsMin} = false$; }
31:       else{//else1
32:          for($j = i + 1; j \le \beta; j++$){//for5
33:             if($P_{\mathrm{last}} = P[j]$) {$\mathrm{IsMin} = false$; break; }
34:          else if$\left(0 < \mathcal{R}_{j-1}^2 \le \mathcal{R}_j^2 - \boldsymbol{\varepsilon}[\varepsilon_{\text{layer}}[j]][j]$ & $P[j] \ge \text{P}_{\text{final}}\right)${
35:             if$\left(\frac{N_{\text{last}}-N[i][\beta]}{P_{\text{last}}-P[i]} > \frac{N_{\text{last}}-N[j][\beta]}{P_{\text{last}}-P[j]}\right)${
36:                IsMin $= false$; break; }}}}//end else1
37:    if(IsMin $= true$){//idxMin $= i$
38:       $\text{more}_{\text{Improve}} = true; t = i;$
39:       $\mathcal{R}_i^2 = \mathcal{R}_i^2 - \boldsymbol{\varepsilon}[\varepsilon_{\text{layer}}[i]][i]$; $\varepsilon_{\text{layer}}[i] + +;$
40:       if$\left(\varepsilon_{\text{layer}}[i] = \ell_{\text{next}}\right)$ {$\boldsymbol{\varepsilon}[\ell_{\text{next}}][1 \ldots \beta] \leftarrow$ Generate_$\varepsilon_{\text{array}}(\beta, \text{P}_{\text{low}}, \mathcal{R}_{\text{low}}, \text{P}_{\text{top}}, \mathcal{R}_{\text{top}}, \mathcal{R}_{\boldsymbol{\varepsilon}}, \text{err}_{\text{layer}}, \ell_{\text{next}})$; $\ell_{\text{next}} + +;$}
41:       if$(2 \times \lceil i/2 \rceil = i)${//i.e., whether $i$ is even or not
42:          $p_{succ}2\left(\mathcal{L}_t, \mathcal{R}_{[1\ldots t]}, [\text{C}, \text{rw}, i]\right)$;//just for update C
43:          $N_{\text{Min}}[t] = N[i][t]$; $t + +;$}//if
44:       for(; $t < \beta$;){//for6
45:          if$(2 \times \lceil i/2 \rceil = i)$ $p_{succ}2\left(\mathcal{L}_{t+1}, \mathcal{R}_{[1\ldots t+1]}, [\text{C}, \text{rw}, i]\right)$;
46:          else $p_{succ}2\left(\mathcal{L}_{t+1}, \mathcal{R}_{[1\ldots t+1]}, [\text{C}, \text{rw}, i+1]\right)$; //Here $p_{succ}2$ is applied just for update C
47:          $N_{\text{Min}}[t] = N[i][t]$; $t + +;$
48:          $N_{\text{Min}}[t] = N[i][t]$; $t + +;$}//end for6
49:       $P_{\text{last}} = P[i]$; $N_{\text{last}} = N[i][\beta]$; $break$; }}//end for4
50: }//end while1
```
Output: optimal bounding function $\mathcal{R}_{opt} = \mathcal{R}$.

Assumption 3 [Average-case]. Uniform selection of index S_{id} in the range of $[1\ to\ \beta - 1]$ to decrease $\mathcal{R}_{i=\text{S}_{\text{id}}}$ by the step of $\boldsymbol{\varepsilon}[\varepsilon_{\text{layer}}[\text{S}_{\text{id}}]][\text{S}_{\text{id}}]$ (see line 39 from Algorithm 8).

By Assumption 3:

$$\text{Avegrage}_{\text{S}_{\text{id}}=1}^{\beta-1}\left[\sum_{t=\left\lceil\frac{\text{S}_{\text{id}}}{2}\right\rceil}^{\frac{\beta}{2}}\left(\frac{(2t)^2\text{S}_{\text{id}}}{8}\right) \times \mathcal{C}\right] \approx \frac{1}{\beta-1}\sum_{\text{S}_{\text{id}}=1}^{\beta-1}\sum_{t=\left\lceil\frac{\text{S}_{\text{id}}}{2}\right\rceil}^{\beta/2}\left(\frac{t^2\text{S}_{\text{id}}}{2}\right) \times \mathcal{C} \approx \left(\frac{3\beta^4+5\beta^3}{480}\right) \times \mathcal{C} \Rightarrow$$

$$\text{T5}_2 \approx (\text{Layer}_{\text{Count}} \times \text{T3}) \times \mathcal{C} + \left(\text{step}_{\text{count}} \times \frac{3\beta^5+8\beta^4+5\beta^3+240\,\beta^2}{480}\right) \times \mathcal{C}. \tag{18}$$

Appendix F. Efficient estimator of success probability

Algorithm 9 efficiently estimates the success probability of $\mathcal{R}$ in Algorithm 8.

Algorithm 9 $p_{succ}2$

Input: lattice block $\mathcal{L}_d$, $\mathcal{R} = [\mathcal{R}_1, \ldots, \mathcal{R}_d]$, enum radii R, $[\text{C}, \text{XY}, i]$/∗ C is a matrix storing some intermediate values in computing p_{succ} based on the idea in Algorithm 8 from [6], X shows that C can be read, Y shows that C can be written, and i is index of current modified entry of $\mathcal{R}$ ∗/

```
1:  if$(d = 1)${//note: $\mathcal{L}_{d=1} = \mathcal{L}_{d=1}^* = [b_d^*]$
2:     if$(\|b_d^*\| \le R\ \mathcal{R}_d)$ $p_{succ} = 1$ else $p_{succ} = 0$;}//end if
3:  else{
4:     for$(k = 0,1)${//for1
5:        if(X = "r" & $d \ge i + 2 - k$) {$c \leftarrow \text{C}[d, i + 2 - k]; j = i;$}
6:        else {$c = 1; j = d;$}
7:        for(; $j \ge 2$; $j -= 2$){//for2
8:           $c = \int_0^x c(t)\, dt$;
9:           $c = c\left(\mathcal{R}_{j-k}^2\right) - c(x)$;
10:          if(Y = "w") {$\text{C}[d, j - k] \leftarrow c$;}}//end for2
11:       $p_k = c(0) \times \left(\frac{d}{2}\right)!$;}//end for1
12:    $\text{P} = \frac{p_0+p_1}{2}$;}//end else
```
Output: success probability P.

Time complexity of p_{succ} in Algorithm 8 from [6] and $p_{succ}2$ (Algorithm 9) with parameters of XY = " − w" or XY = " − −" is determined as follows (d is partial block size in range of $1 \le d \le \beta$):

$$\text{T6}_1 \approx \sum_{k=0}^{1}\sum_{j=1}^{d/2}\left(\sum_{l=1}^{d/2}(1) \times \mathcal{C} + \sum_{l=1}^{d/2}\sum_{t=2}^{l}(1) \times \mathcal{C}\right) \approx \sum_{k=0}^{1}\sum_{j=1}^{d/2}\sum_{l=1}^{d/2}\sum_{t=2}^{l}(1) \times \mathcal{C} \approx \left(\frac{d^3}{8}\right) \times \mathcal{C}. \tag{19}$$

Time complexity of $p_{succ}2$ in Algorithm 9 with parameters of XY = "r − " or XY = "rw" is determined as follows (input parameter i is defind in Algorithm 9):

$$\text{T6}_2 \approx \sum_{k=0}^{1}\sum_{j=1}^{\frac{i}{2}}\left(\sum_{l=1}^{\frac{d}{2}}(1) \times \mathcal{C} + \sum_{l=1}^{\frac{d}{2}}\sum_{t=2}^{l}(1) \times \mathcal{C}\right) \approx \sum_{k=0}^{1}\sum_{j=1}^{\frac{i}{2}}\sum_{l=1}^{\frac{d}{2}}\sum_{t=2}^{l}(1) \times \mathcal{C} \approx \left(\frac{d^2 i}{8}\right) \times \mathcal{C}. \tag{20}$$

Appendix G. Final phase of generating $\mathcal{R}_{\mathrm{A}}$ in Algorithm 3

Algorithm 10 is the final phase of generating $\mathcal{R}_{\mathrm{A}}$ in Algorithm 3 (Approximate_$\mathcal{R}_{\mathrm{opt}}$_1).

Algorithm 10 Generate_$\mathcal{R}_{\mathrm{opt}}$_4

```
Input: block size $\beta$, $\mathrm{P_{final}}$, initial bounding function $\mathcal{R}$, $\mathcal{R}_{\mathrm{low}}$.
 1:   for$\left(i = 1; i \le \frac{\beta}{2}; i++\right)${//for1
 2:       for($j = 1; j \le i; j++$) {$\mathcal{R}[j] \leftarrow \mathcal{R}_{\mathrm{low}}[i]$; }
 3:       $\mathrm{P} \leftarrow p_{succ}(\mathcal{R})$;
 4:       if($\mathrm{P} \ge \mathrm{P_{final}}$) break;
 5:   }//end for1
Output: Final shape of $\mathcal{R}$ in approximating optimal bounding function
```

Time complexity of Algorithm 10 is determined as follows:

Assumption 4 [Average-case]. Uniform selection of index $i = \mathrm{S_{id}}$ in the range of $[1\ to\ \beta/2]$ by satisfying the condition in line 4 from Algorithm 10.

By using Average-case Assumption 4:

$$\mathrm{T7} \approx \mathrm{Avegrage}_{\mathrm{S_{id}}=1}^{\beta/2} \sum_{i=1}^{\mathrm{S_{id}}} \left(\left(\frac{\beta^3}{8}\right) \times \mathcal{C} + \sum_{j=1}^{i} (1) \times \mathcal{C} \right) \Rightarrow$$

$$\mathrm{T7} \approx \frac{1}{\beta/2} \sum_{\mathrm{S_{id}}=1}^{\beta/2} \sum_{i=1}^{\mathrm{S_{id}}} \left(\left(\frac{\beta^3}{8}\right) \times \mathcal{C} \right) \approx \left(\frac{\beta^4}{32}\right) \times \mathcal{C}. \quad (21)$$

References

[1] D. Micciancio and O. Regev, “Lattice-based cryptography”, In Post-quantum cryptography, pp. 147-191, Springer Berlin Heidelberg, 2009.

[2] Moghissi, Gholam Reza, and Ali Payandeh. "A Parallel Evolutionary Search for Shortest Vector Problem." International Journal of Information Technology and Computer Science, (2019).

[3] Zhongxiang Zheng, Xiaoyun Wang, Yang Yu, “Orthogonalized lattice enumeration for solving SVP”, Science China Information Sciences 2018.

[4] Aono, Yoshinori, Phong Q. Nguyen, Yixin Shen, “Quantum lattice enumeration and tweaking discrete pruning”, International Conference on the Theory and Application of Cryptology and Information Security. Springer, Cham, 2018.

[5] N. Gama, P. Q. Nguyen, and O. Regev, “Lattice enumeration using extreme pruning”, In Proc. EUROCRYPT ’10, volume 6110 of LNCS. Springer, 2010.

[6] Y. Chen, and P. Q. Nguyen, “BKZ 2.0: Better lattice security estimates”, In International Conference on the Theory and Application of Cryptology and Information Security, pp. 1-20. Springer Berlin Heidelberg, 2011.

[7] ACD+18. M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer. “Estimate all the {LWE, NTRU} schemes!”, In SCN, pages 351–357, 2018.

[8] Moghissi, Gholam Reza, and Ali Payandeh. "Using Progressive Success Probabilities for Sound-pruned Enumerations in BKZ Algorithm." International Journal of Computer Network and Information Security 9.9 (2018): 10.

[9] Moghissi, G., Payandeh, A. (2021). “Better Sampling Method of Enumeration Solution for BKZ-Simulation”, *The ISC International Journal of Information Security*, 13(2), pp. 177-208. doi: 10.22042/isecure.2021.225886.531.

[10] Moghissi, Gholam Reza, and Ali Payandeh. “Rejecting claimed speedup of $2^{\beta/2}$ in extreme pruning and revising BKZ 2.0 for better speedup”, Journal of Computing and Security, 2021; 8(1): 65-91. doi: 10.22108/jcs.2021.121191.1044.

[11] “SVP Challenge”, [Online]. Available: https://www.latticechallenge.org/svp-challenge/index.php.

[12] Moghissi, Gholam Reza, and Ali Payandeh. "Formal Verification of NTRUEncrypt Scheme." International Journal of Computer Network and Information Security 8.4 (2016): 44.

Authors’ Profiles

Gholam Reza Moghissi received the M.S. degree in the Department of ICT at Malek-e-Ashtar University of Technology, Tehran, Iran, in 2016. His researches focus on Information Security.

Ali Payandeh received the M.S. degree in Electrical Engineering from Tarbiat Modares University in 1994, and the Ph.D. degree in Electrical Engineering from K.N. Toosi University of Technology (Tehran, Iran) in 2006. He is now an assistant professor in the Department of Information and Communications Technology at the Malek-e-Ashtar University of Technology, Iran. He has published many papers in international journals and conferences. His research interests include information theory, coding theory, cryptography, security protocols, secure communications, and satellite communications.

How to cite this paper: Gholam Reza Moghissi, Ali Payandeh," Optimal bounding function for GNR-enumeration ", International Journal of Mathematical Sciences and Computing(IJMSC), Vol.8, No.1, pp. 1-17, 2022. DOI: 10.5815/ijmsc.2022.01.01