A Report on an Advanced Sieving Algorithm with DualBasis centers and Voronoi Cells

August 27, 2025

Abstract

In the original article, when the sieve method is applied, the centers of the filters are randomly selected, which exhibited a high degree of randomness and was non-reproducible. Moreover, the filtration condition involves a large amount of overlapping regions in the spherical caps. Both factors lead to a waste of computing power. Therefore, two feasible improvement methods are proposed: using the dual lattice and the enumerated low-norm vectors as sieve centers, and adopting a purely algebraic method for filtration. This approach is reproducible and can optimize the computing power requirements to a certain extent.

1 Introduction and Motivation

Sieving algorithms are fundamental tools in lattice-based cryptography, primarily used for solving the Shortest Vector Problem (SVP). Their performance is critical for the practical security analysis of lattice-based schemes. The bgj3 algorithm (Algorithm 1) represents a significant milestone, but its reliance on random projections introduces several challenges:

- **Probabilistic Nature:** Non-reproducible, the probability of finding good vector pairs depends on random selection.
- **Redundant Computations:** Randomly chosen spherical caps can heavily overlap, causing the same vector pairs to be tested repeatedly.
- **Inefficiency:** The process is agnostic to the underlying algebraic structure of the lattice, missing key optimization opportunities.

To address these limitations, we propose a new hybrid algorithm that is deterministic, structurally aware, and designed for efficient implementation. The following is the original algorithm.

Algorithm 1 AllPairSearch - bgj3 (Baseline)

```
Require: A list $L$ of $N_0$ lattice vectors, repetitions $(B_0, B_1, B_2)$, radii $(\alpha_0, \alpha_1, \alpha_2)$, goal norm $l$.
Ensure: A list of reducing pairs in $L$.
 1: $\mathcal{N} \leftarrow \emptyset$
 2: for $i = 0, \ldots, B_0 - 1$ do
 3:     Pick a random center $c_0$ from $S^{n-1}$
 4:     Compute $L_i \leftarrow \{v \in L \mid v$ passes $F_{c_0,\alpha_0}\}$
 5:     for $j = 0, \ldots, B_1/B_0 - 1$ do
 6:         Pick a random center $c_1$ from $S^{n-1}$
 7:         $L_{ij} \leftarrow \{v \in L_i \mid v$ passes $F_{c_1,\alpha_1}\}$
 8:         for $k = 0, \ldots, B_2/B_1 - 1$ do
 9:             Pick a random center $c_2$ from $S^{n-1}$
10:             $L_{ijk} \leftarrow \{v \in L_{ij} \mid v$ passes $F_{c_2,\alpha_2}\}$
11:             $\mathcal{N} \leftarrow \mathcal{N} \cup \{(u, v) \in L_{ijk}^2 \mid \|u \pm v\| < l\}$
12:         end for
13:     end for
14: end for
15: return $\mathcal{N}$
```

2 Design Philosophy and Evolution

The core of our new approach is to replace randomness with deterministic, structurally-informed choices.

2.1 The Role of the Dual Lattice

The dual lattice $\mathcal{L}(\mathbf{B}^\vee)$ provides a powerful analytical tool. For a vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ and a dual vector $\mathbf{u} \in \mathcal{L}(\mathbf{B}^\vee)$, their inner product $\langle \mathbf{v}, \mathbf{u} \rangle$ is always an integer. This property allows us to define sharp, algebraic filters instead of fuzzy, geometric ones. If two vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ are close, i.e., $\|\mathbf{v}_a - \mathbf{v}_b\| \to 0$, then for any dual vector $\mathbf{u}$, the difference in their inner products $\langle \mathbf{v}_a, \mathbf{u} \rangle - \langle \mathbf{v}_b, \mathbf{u} \rangle = \langle \mathbf{v}_a - \mathbf{v}_b, \mathbf{u} \rangle$ will be a small integer. This insight is the foundation of our coarse-sieving phase.

2.2 Evolution of Center Generation

A critical component of advanced sieve is the set of center vectors used for partitioning. Our design evolved through the following stages:

1. **Initial Idea (Linear Combinations):** My first thought was to generate a rich set of center vectors by taking short integer linear combinations of the dual basis vectors $\mathbf{b}_i^\vee$. This approach is flexible and can theoretically produce a dense set of centers.
2. **Encountered Problem (Computational Cost):** However, generating, managing, and ensuring the quality (i.e., short norm and good angular distribution) of these combinations on-the-fly was computationally intensive. It introduced significant overhead that probably threatened to negate the benefits of the deterministic approach.
3. **Final Strategy (Enumeration):** I pivoted to a more stable and efficient method: enumerating a large pool of the shortest dual lattice vectors up to a certain norm bound. This is achieved using a standard lattice enumeration algorithm (e.g. KFP, SE,...). This method has several advantages:
   - It guarantees that we obtain the absolute shortest vectors in all directions.
   - The generation is a one-time cost(although it requires complex calculation).
   - The resulting pool of vectors can be sorted by norm, providing a ready-made, high-quality source of centers for the subsequent partitioning phase.

This evolution led to the robust, two-phase structure of the final algorithm.

3 The Hybrid Sieve Algorithm

The final algorithm (Algorithm 2) integrates the insights above.

Algorithm 2 AllPairSearch with Hybrid Sieve

```
Require: The LLL-reduced basis $\mathbf{B}$; A list $L_0$ of vectors; A goal norm $\ell$; Number of repetitions for
    each level $(B_0, B_1, B_2)$; Total number of enumerated centers $N_{\text{enum-c}}$; Coarse-sieve integer range
    $K_{\text{rangeCoarse}}$.
Ensure: A list $N_{out}$ of reducing pairs.
 1: $N_{out} \leftarrow \emptyset$

    Phase 1: Preparation
 2: $\mathbf{B}^\vee \leftarrow \text{DualBasis}(\mathbf{B})$
 3: $R_{enum\_c} = 1.2\lambda_1(\mathcal{L}(B^\vee))$
 4: $C_{\text{dense\_pool}} \leftarrow \text{EnumerateShortDualVectors}(\mathbf{B}^\vee, N_{\text{enum-c}}, R_{enum\_c})$
 5: Sort $C_{\text{dense\_pool}}$ by increasing norm.

    Phase 3: Hierarchical Partitioning with Interleaved Centers
 6: $K \leftarrow 3$                                  ▷ The number of hierarchical levels
 7: $num\_c0 \leftarrow B_0$
 8: $num\_c1 \leftarrow B_1/B_0$
 9: $num\_c2 \leftarrow B_2/B_1$
10: $N_{\text{total\_centers}} \leftarrow num\_c0 + num\_c1 + num\_c2$

11: $C_0, C_1, C_2 \leftarrow \emptyset$
12: for $i \leftarrow 0$ to $N_{\text{total\_centers}} - 1$ do
13:     target_layer $\leftarrow i \pmod K$               ▷ Deal centers like dealing cards
14:     if target_layer $== 0$ and $|C_0| < num\_c0$ then
15:         $C_0 \leftarrow C_0 \cup \{C_{\text{dense\_pool}}[i]\}$
16:     else if target_layer $== 1$ and $|C_1| < num\_c1$ then
17:         $C_1 \leftarrow C_1 \cup \{C_{\text{dense\_pool}}[i]\}$
18:     else if target_layer $== 2$ and $|C_2| < num\_c2$ then
19:         $C_2 \leftarrow C_2 \cup \{C_{\text{dense\_pool}}[i]\}$
20:     end if
21: end for

22: for each $\mathbf{c}_0 \in C_0$ do
23:     $L_i \leftarrow \{\mathbf{v} \in L' \mid \text{FindClosestVector}(\mathbf{v}, C_0) == \mathbf{c}_0\}$
24:     if $|L_i| \leq 1$ then continue
25:     end if
26:     for each $\mathbf{c}_1 \in C_1$ do
27:         $L_{ij} \leftarrow \{\mathbf{v} \in L_i \mid \text{FindClosestVector}(\mathbf{v}, C_1) == \mathbf{c}_1\}$
28:         if $|L_{ij}| \leq 1$ then continue
29:         end if
30:         for each $\mathbf{c}_2 \in C_2$ do
31:             $L_{ijk} \leftarrow \{\mathbf{v} \in L_{ij} \mid \text{FindClosestVector}(\mathbf{v}, C_2) == \mathbf{c}_2\}$
32:             if $|L_{ijk}| > 1$ then
33:                 $N_{out} \leftarrow N_{out} \cup \{(u, v) \in L_{ijk}^2 \mid u \neq v, \|u \pm v\| < l\}$
34:             end if
35:         end for
36:     end for
37: end for
38: return $N_{out}$
```

4 Key Advantages and Discussion

The hybrid algorithm offers significant advantages over traditional methods:

1. **Deterministic and Reproducible:** The algorithm is fully deterministic. Given the same input, it will always produce the same output, which is quite valuable for complexity analysis and debugging.

2. **Structurally Informed:** The coarse sieve (Phase 2) leverages the fundamental algebraic prop-

erties of the lattice, filtering from a ”structural perspective” to retain only vectors that lie near specific hyperplanes defined by the shortest dual vectors. This is far more effective reducing ”bad vectors”.

3. **Efficient Partitioning:** The hierarchical filtering (Phase 3) uses a finite set of pre-computed, high-quality centers. The partitioning function, ‘FindClosestVector‘, is a simple linear scan over a small set of centers, having a low complexity of $O(B_i \cdot n)$.
4. **Non-overlapping Partitions:** The Voronoi-cell-based partitioning ensures that each vector is assigned to exactly one bucket at each level. This completely eliminates the redundant computations that plague the overlapping spherical caps of the bgj3 algorithm.
5. **Parameter Cohesion:** A key design feature is the direct link between the repetition parameters (B_0, B_1, B_2) and the number of centers used at each level. The vectors for partitioning are allocated to each layer all at once during the preparation phase, making the algorithm's behavior clear and controllable.

4.1 Implementation Note: Handling Overlapping Vectors

In the partitioning phase, it's possible for a vector to be share the same distance to two or more centers in a set C_i. A deterministic tie-breaking rule (e.g. choosing the center with smaller lexicographical order) is necessary to ensure each vector is assigned to a unique bucket.

5 Conclusion

The Hybrid Sieve algorithm could be a step forward from probabilistic sieving techniques. By systematically replacing randomness with deterministic, structurally-aware mechanisms, it achieves superior efficiency and control. The final algorithm presents a powerful and practical framework for solving the Shortest Vector Problem(SVP), promising substantial performance improvements in real-world cryptographic analysis.

A Conceptual Helper Functions

- **DualBasis($\mathbf{B}$):** Computes the basis $\mathbf{B}^\vee$ for the dual lattice $\mathcal{L}(\mathbf{B}^\vee)$.
- **EnumerateShortDualVectors($\mathbf{B}^\vee, N_{\textbf{enum-c}}, R_{enum\_c}$):** Runs a lattice enumeration algorithm to find the $N_{\text{enum-c}}$ shortest non-zero vectors in $\mathcal{L}(\mathbf{B}^\vee)$.
- **FindClosestVector($\mathbf{v}, C_{\textbf{set}}$):** Finds the vector in the **finite set** C_{set} that is closest to $\mathbf{v}$ (in Euclidean distance). This is a simple linear scan over the elements of C_{set}, with a deterministic tie-breaking rule.