Vol.2021-DPS-186 No.68
Vol.2021-CSEC-92 No.68
2021/3/16

General Sieve Kernelの考察および改良

長谷川 奨[1]　王 イントウ[1]　藤崎 英一郎[1]

概要：2019 年に Albrecht 等が提案した General Sieve Kernel(G6K) は，2021 年 2 月時点のダルムシュタット工科大学が主催している SVP Challenge にて 180 次元の近似最短ベクトルを見つけている．使用時間の制限がある大規模計算機と G6K を使用して近似最短ベクトルを見つける場合，近似最短ベクトルが見つかる前にプログラムが停止し，解読データが削除されてしまう．本稿では，G6K に保存機能の追加とパラメータの調整を行った．その結果，SVP Challenge で未解読次元であった 154，156，158 次元の近似最短ベクトルを見つけた．

1. はじめに

現在，大規模な量子計算機の開発が盛んになっている．それにともない，素因数分解の解読困難性や楕円曲線上の離散対数問題を安全性の根拠としている公開鍵暗号は Shor のアルゴリズム [1] を使用することにより多項式時間で解読されることが知られている．NIST は，大規模な量子計算機でも解読が困難な暗号方式の標準化に力をいれている [2]．2020 年，7 月には 3 回目の選定である Round3 の発表があった．公開鍵暗号では 4 つの暗号方式が残り，その中でも格子の性質を利用した暗号方式が 3 つ存在している．デジタル署名方式でも，3 件中 2 件が格子の性質を使用している．このように，耐量子計算機暗号の候補として格子暗号が最有力候補となっている．

格子暗号の安全性は，最短ベクトル問題 (SVP) や最近ベクトル問題 (CVP) などの格子の数学的問題を安全性の根拠としている．現時点では，SVP や CVP を多項式時間で解くことができるアルゴリズムは存在していない．そこで，条件を緩くした近似 SVP や近似 CVP を解くことができるアルゴリズムを使用して安全性の検証を行っている．近似 SVP は，LLL 基底簡約アルゴリズム [3] や BKZ 基底簡約アルゴリズム [4] のような基底簡約アルゴリズムや，Sieve[5] や ENUM[4] といった格子点探索アルゴリズムによって解くことができる．ドイツのダルムシュタット工科大学が主催している SVP Challenge[6] で，高次元格子の近似最短ベクトルの解読に成功しているアルゴリズムは，Sieve や BKZ 基底簡約アルゴリズムに分類されるアルゴリズムたちである [7][8][9]．Sieve は，2001 年に Ajtai らが提案し，Nguyen らが実装実験を行った [10]．今日に至るまで様々な改良がされてきた．2019 年には，Albrecht らにより，General Sieve Kernel(G6K)[7] が提案され，2021 年 2 月時点の SVP Challenge にて 180 次元の近似最短ベクトルを見つけている．

G6K は，格子の近似最短ベクトルを見つけるために G6K 内の Pump アルゴリズムと WorkOut アルゴリズムを使用する．時間などの使用制限がある大規模計算機と G6K を使用して近似最短ベクトルを見つける場合，時間制限などでプログラムが途中で停止してしまい，それまでに計算したデータが消えてしまう．本稿では，プログラムが途中で停止しても，解読に必要なデータが保存されるように G6K を改良した．また，WorkOut アルゴリズム実行時に Sieve を行う範囲の増加値の最適な値を実験により決めた．最後に，改良した G6K と実験により決めた値を使用して，SVP Challenge の未解読次元である 154，156，158 次元格子の近似最短ベクトルを見つけた結果を載せる．

2. 準備

2.1 格子

ベクトル空間 $\mathbb{R}^m$ の n 個のベクトル $\mathbf{b}_1, \ldots, \mathbf{b}_n$ の整数係数の線形結合全体の集合を

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) := \left\{ \sum_{i=1}^{n} a_i \mathbf{b}_i \in \mathbb{R}^m : a_i \in \mathbb{Z} \right\}$$

と定義する．

一次独立なベクトル $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$ の整数係数の線型結合全体の集合 $\mathcal{L} = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ を $\mathbb{R}^m$ の格子と定義する．また，格子 $\mathcal{L}$ が生成する一次独立な n 個のベクトルの組 $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ を基底，各 $\mathbf{b}_i$ を基底ベクトルと定義

[1] 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology
石川県能美市旭台 1-1，923-1292

する．

異なる基底で同じ格子が生成されるとき，格子の体積は不変である．n 次元格子 $\mathcal{L}$ の基底 $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ としたときの格子の体積を，

$$\mathrm{vol}(\mathcal{L}) := \sqrt{\Delta(\mathbf{b}_1, \ldots, \mathbf{b}_n)}$$

と定義する．特に $n = m$ の場合，格子 $\mathcal{L}$ の任意の基底行列 $\mathbf{B}$ に対して

$$\mathrm{vol}(\mathcal{L}) = \sqrt{(\det(\mathbf{B})\det(\mathbf{B}^\top))} = |\det(\mathbf{B})|$$

が成り立つ．

2.2 Gram-Schmidt の直交化

n 次元格子の順序付き基底 $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ に対する Gram-Schmidt 直交化 (GSO) ベクトル $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^* \in \mathbb{R}^m$ と GSO 係数 $\mu_{i,j}$ を

$$\begin{cases} \mathbf{b}_1^* := \mathbf{b}_1, \\ \mathbf{b}_i^* := \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^* \ (2 \le i \le n). \end{cases}$$

$$\mu_{i,j} := \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{||\mathbf{b}_j^*||^2} \ (1 \le j < i \le n)$$

と定義する．

GSO ベクトル $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$ は基底 $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ の順序に依存する．また，GSO ベクトル $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$ は格子の基底にならない．

2.3 射影格子

n 次元格子 $\mathcal{L} \subseteq \mathbb{R}^m$ の基底を $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ とし，各 $1 \le l \le n$ に対して，ベクトル空間 $\mathbb{R}^m$ から $\mathbb{R}$ ベクトル空間 $\langle \mathbf{b}_1, \ldots, \mathbf{b}_{l-1} \rangle_{\mathbb{R}}$ の直交補空間への直交射影を

$$\pi_l : \mathbb{R}^m \longrightarrow \langle \mathbf{b}_1, \ldots, \mathbf{b}_{l-1} \rangle_{\mathbb{R}}^{\perp}$$

と定義する．ただし，π_1 は恒等写像．

また，任意のベクトル $\mathbf{x} \in \mathrm{span}_{\mathbb{R}}(\mathcal{L})$ に対して，

$$\pi_l(\mathbf{x}) = \sum_{j=l}^{n} \frac{\langle \mathbf{x}, \mathbf{b}_j^* \rangle}{||\mathbf{b}_j^*||^2} \mathbf{b}_j^*$$

が成り立つ．集合 $\pi_l(\mathcal{L})$ は一次独立なベクトル $\pi_l(\mathbf{b}_l), \cdots, \pi_l(\mathbf{b}_n)$ の整数係数結合全体と一致するので，集合 $\pi_l(\mathcal{L})$ は

$$\{\pi_l(\mathbf{b}_l), \ldots, \pi_l(\mathbf{b}_n)\}$$

を基底に持つ $(n - l + 1)$ 次元の格子になる．

また，$\mathcal{L}_{[l:r]}$ と $\mathbf{B}_{[l:r]}$ をそれぞれ

$$\mathcal{L}_{[l:r]} := \pi_l(\mathcal{L}(\mathbf{b}_l)), \ldots, \pi_l(\mathcal{L}(\mathbf{b}_r))$$

$$\mathbf{B}_{[l:r]} := \pi_l(\mathbf{b}_l), \ldots, \pi_l(\mathbf{b}_r)$$

と定義する．

2.4 逐次最小

n 次元格子 $\mathcal{L}$ に対して，各 $1 \le i \le n$ における逐次最小を，

$$\lambda_i(\mathcal{L}) := \min_{\mathbf{b}_1, \ldots, \mathbf{b}_i \in L} \max\{||\mathbf{b}_1||, \ldots, ||\mathbf{b}_i||\}$$

と定義する．ただし，格子ベクトル $\mathbf{b}_1, \ldots, \mathbf{b}_i \in L$ は一次独立とする．

定義 1. n 次元格子 $\mathcal{L}$ において，すべての $1 \le i \le n$ に対し $||\mathbf{b}_i|| = \lambda_i(\mathcal{L})$ を満たす一次独立な格子ベクトル $\mathbf{b}_1, \cdots, \mathbf{b}_n \in \mathcal{L}$ を格子 $\mathcal{L}$ の逐次最小ベクトルと呼ぶ．さらに，そのベクトルの組が格子 $\mathcal{L}$ の基底になるとき，そのベクトルの組を逐次最小基底と呼ぶ．

2.5 Gaussian heuristic

Gaussian heuristic とは，$\mathbb{R}^n$ 内の完全階数の格子 $\mathcal{L}$ に対して，体積を持つ任意の集合 $C \subseteq \mathbb{R}^n$ との共通部分 $\mathcal{L} \cap C$ に含まれる格子ベクトルの個数 $\#(\mathcal{L} \cap C)$ はおおよそ $\mathrm{vol}(C)/\mathrm{vol}(\mathcal{L})$ であると期待できることをいう．特に，集合 C を 0 を中心とした半径が逐次最小ベクトル $\lambda_1(\mathcal{L})$ の n 次元球とすると

$$\frac{\mathrm{vol}(C)}{\mathrm{vol}(\mathcal{L})} \approx \#(\mathcal{L} \cap C) \approx 1$$

が期待できる．さらに，$\mathrm{vol}(C) = \frac{\pi^{n/2}}{\Gamma(1+n/2)}\lambda_1(\mathcal{L})^n$ より，逐次最小ノルムは Gaussian heuristic を用いることにより

$$\begin{aligned} \lambda_1(\mathcal{L}) &\approx \left(\frac{\Gamma(1+\frac{n}{2})\mathrm{vol}(\mathcal{L})}{\pi^{n/2}}\right)^{\frac{1}{n}} \\ &\sim \sqrt{\frac{n}{2\pi e}}\mathrm{vol}(\mathcal{L})^{\frac{1}{n}} \\ &= \mathrm{GH}(\mathcal{L}) \end{aligned}$$

が期待できる．

2.6 最短ベクトル問題 (SVP)

n 次元格子 $\mathcal{L} \subseteq \mathbb{Z}^m$ の基底 $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ が与えられた時，

$$||\mathbf{v}|| = \lambda_1(\mathcal{L})$$

を満たす格子上の非零ベクトル $\mathbf{v} \in \mathcal{L}$ を見つける問題が最短ベクトル問題である．

近似因子 $\gamma \ge 1$ を用いることで，$\gamma \cdot \mathrm{GH}(\mathcal{L})$ 以下のノルムを最短ベクトルとする問題を近似最短ベクトル問題と呼ぶ．

2.7 SVP Challenge[6]

SVP Challenge はドイツのダルムシュタット工科大学が開設，運用を行っている．ここでは，近似最短ベクトルの近似因子 $\gamma = 1.05$ 以下のノルムを最短ベクトルとしており，各次元の格子の基底が具体的に提供されている．世界中の研究者たちは生成される基底を用いて，アルゴリズムの性能の比較や検証を行っている．2021 年 2 月までに，180 次元格子の近似最短ベクトルが求められている．

2.8 LLL 基底簡約アルゴリズム [3]

LLL 基底簡約アルゴリズムは，格子 $\mathcal{L}$ 上の第一逐次最小 $\lambda_1(\mathcal{L})$ にある指数近似因子をかけた値より小さいノルムを持つ格子ベクトルを効率的に見つけるアルゴリズムである．

定義 2. n 次元格子 $\mathcal{L}$ の基底 $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ の GSO ベクトルを $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$ と GSO 係数 $\mu_{i,j}(1 \leq j < i \leq n)$ とし，簡約パラメータ $\frac{1}{4} < \delta < 1$ に関する次の 2 つの条件を満たすとき，その基底は δ に関して LLL 簡約されている．

(1) 基底 $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ の GSO 係数 $\mu_{i,j}$ が $|\mu_{i,j}| \leq \frac{1}{2}(1 \leq \forall j < \forall i \leq n)$ を満たす．

(2) 任意の $2 \leq k \leq n$ に対して，$\delta||\mathbf{b}_{k-1}^*||^2 \leq ||\pi_{k-1}(\mathbf{b}_k)||^2$ を満たす (Lovász 条件)．

LLL 基底簡約アルゴリズムを **Algorithm 1** に示す．

Algorithm 1 LLL 基底簡約アルゴリズム:LLL$(\mathbf{B}, \delta)$

Input: n 次元格子の基底 $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$，パラメータ $\frac{1}{4} < \delta < 1$

Output: 格子 $\mathcal{L}$ の δ に関する LLL 基底簡約された基底 $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$

```
入力基底の GSO ベクトル {b*_1, ··· , b*_n} と GSO 係数 μ_{i,j}(1 ≤ j < i ≤ n) を計算
B_i ← ||b*_i||^2 (1 ≤ i ≤ n)
k ← 2
while k ≤ n do
    for j = k − 1 down to 1 do
        if |μ_{i,j}| > 1/2 then
            q ← ⌊μ_{i,j}⌉, b_i ← b_i − q b_j
            for l = 1 to j do
                μ_{i,l} ← μ_{i,l} − q μ_{j,l}
            end for
        end if
    end for
    if B_k ≥ (δ − μ^2_{k,k−1}) B_{k−1} then
        k ← k + 1
    else
        swap(b_{k−1}, b_k)
        GSO 情報の更新
        k ← max{k − 1, 2}
    end if
end while
```

2.9 Babai's nearest plane アルゴリズム [11]

目標ベクトルにもっとも近い格子ベクトルを見つける代表的なアルゴリズムは Babai's nearest plane アルゴリズムである．格子 $\mathcal{L}$ の基底 $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ と目標ベクトル $\mathbf{w}$ が入力された時に，最近ベクトル $\mathbf{v} \in \mathcal{L}$ を見つけるアルゴリズムである．Babai's nearest plane アルゴリズムを **Algorithm 2** に示す．

Algorithm 2 Babai's nearest plane アルゴリズム:Babai$(\mathbf{B}, \mathbf{w})$

Input: n 次元格子の基底 $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$，目標ベクトル $\mathbf{w} \in \mathbb{Z}^n$

Output: 目標ベクトル $\mathbf{w}$ に近い格子ベクトル $\mathbf{v} \in \mathcal{L}$

```
入力基底の GSO ベクトル {b*_1, ..., b*_n} を計算
b ← w
for i = n down to 1 do
    l = ⟨b, b*_i⟩ / ||b*_i||^2 ∈ ℚ の最近似整数 c = ⌊l⌉ の計算
    b ← b − c b_i
end for
return v = w − b ∈ L
```

3. 関連研究

3.1 Sieve[5]

Sieve アルゴリズムは，ベクトル同士を引き算していくことで最短ベクトルを求めるアルゴリズムである．具体的は，以下の 4 ステップを繰り返す．

(1) 入力された n 次元格子の基底を使用して 2^n 個の格子点をサンプリングする．

(2) サンプリングされた格子点の中でノルムが一番大きい格子点を半径 R_0 の球で格子点を覆う．

(3) 半径 $R_1 < R_0$ の複数の球でサンプリングされたすべての格子点を覆う．

(4) 半径 R_1 の球の中心点を原点へ平行移動する．

この 4 ステップの処理を繰り返すことにより入力された格子の最短ベクトルを見つける．Sieve アルゴリズムを **Algorithm 3** に示す．

Algorithm 3 Sieve:Sieve$(\mathbf{B})$

Input: n 次元格子 $\mathcal{L}$ の基底 $\mathbf{B} = \{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$

Output: ノルムが $\sqrt{4/3}\mathrm{gh}(\mathcal{L})$ 以下のベクトルのリスト L

```
L ← a set of N random vectors (of length at most 2^n Vol(L)^{1/n}) from L where N = (4/3)^{n/2+o(n)}
while ∃(v, w) ∈ L^2 such that ||v − w|| < ||v|| do
    v ← v − w
end while
return L
```

3.2 SubSieve アルゴリズム [12]

SubSieve アルゴリズムは，Sieve アルゴリズムが $\sqrt{4/3}\mathrm{gh}(\mathcal{L})$ 以下のノルムの格子ベクトルを多く生成することを利用している．n 次元格子 $\mathcal{L}$ と $d \approx n \cdot \ln(4/3)/\ln(n/2\pi e)$ である $n-d$ より小さい次元の射影された部分格子で Sieve アルゴリズムと Babai's nearest plane アルゴリズムを使用することにより n 次元格子の SVP を解く．

射影された $n-d$ 次元の部分格子 $\mathcal{L}_d$ に対し Sieve アルゴリズムを行った結果をリスト

$$L := \mathsf{Sieve}(\mathcal{L}_d) = \{\mathbf{x} \in \mathcal{L}_d \setminus \{\mathbf{0}\} \mid ||\mathbf{x}|| \leq \sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}_d)\} \quad (1)$$

に格納する．ここで，n 次元格子上の最短ベクトルを $\mathbf{s} = \mathbf{Bx}$ の格子の基底と整数係数で表す．整数係数 $\mathbf{x}$ を $\mathbf{x}' \in \mathbb{Z}^d$ と $\mathbf{x}'' \in \mathbb{Z}^{n-d}$ に分割し，$n-d$ 次元の部分射影格子 $\mathcal{L}_d$ 上の最短ベクトル $\mathbf{s}_d = \pi_d(\mathbf{Bx}) = \mathbf{B}_d\mathbf{x}''$ に近い d 次元部分格子上のベクトルを Babai's nearest plane アルゴリズムを使用し見つける．その後，見つけた d 次元部分格子上のベクトルと $n-d$ 次元部分射影格子上の最短ベクトルを加算することにより n 次元格子 $\mathcal{L}$ 上の最短ベクトルを復元する．

SubSieve アルゴリズムを **Algorithm 4** に示す．

Algorithm 4 SubSieve アルゴリズム:$\mathsf{SSieve}(\mathbf{B})$

Input: n 次元格子 $\mathcal{L}$ の基底 $\mathbf{B} = [\mathbf{B}'|\mathbf{B}'']$
Output: 格子 $\mathcal{L}$ の最短ベクトル $\mathbf{v} \in \mathcal{L}(\mathbf{B})$

```
L ← Sieve(L_d)
for each w_i ∈ L do
    Compute x''_i such that B_d · x''_i = w_i
    t_i = B'' · x''
    s_i ← Babai(B', t_i) + t_i
end for
return the shortest s_i
```

SubSieve アルゴリズムは，部分射影格子の次元を固定した状態で実行していた．SubSieve$^+$ では，SubSieve が終了するごとにインデックス 0 から $n/2-1$ までの範囲の基底を SubSieve で出力された線形独立な基底と交換し，基底全体に LLL 基底簡約アルゴリズムを実行し，d の値を増加させる．この処理を最短ベクトルが見つかるか $d = n$ になるまで繰り返す．ここでいう $d = n$ とは，n 次元の格子に対して Sieve を実行したことと同じである．最短ベクトルが出力されなくとも，各ステップで HKZ 簡約されているので全体の基底は良くなっていっていく．

SubSieve$^+$ アルゴリズムを **Algorithm 5** に示す．

Algorithm 5 SubSieve$^+$ アルゴリズム:$\mathsf{S^+Sieve}(\mathbf{B})$

Input: n 次元格子 $\mathcal{L}$ の基底 $\mathbf{B} = [\mathbf{B}'|\mathbf{B}'']$
Output: 基底簡約された基底 $\{\mathbf{v}_1, \cdots, \mathbf{v}_{n/2}, \mathbf{b}_{n/2+1}, \cdots, \mathbf{b}_n\}$

```
L ← Sieve(L_d)
for each w_i ∈ L do
    Compute x''_i such that B_d · x''_i = w_i
    t_i = B'' · x''
    s_i ← Babai(B', t_i) + t_i
end for
for j = 1, ··· , n/2 do
    Set v_j to be the s_i vector minimizing ||π_(v_1,···,v_j)^⊥(s_i)|| such that s ∉ Span(v_1, ··· , v_j)
end for
return (v_1, ··· , v_{n/2}, b_{n/2+1}, ··· , b_n)
```

3.3 General Sieve Kernel(G6K)[7]

General Sieve Kernel(G6K) は，格子の近似最短ベクトルを求めるために内部にある Pump アルゴリズムと WorkOut アルゴリズムを使用する．Pump アルゴリズムは，Sieve アルゴリズムと復元アルゴリズムを使用した Pump という処理を繰り返すアルゴリズムである．WorkOut アルゴリズムは Sieve を行う範囲を増加させながら Pump をアルゴリズムを近似最短ベクトルが見つかるまで繰り返すアルゴリズムである．Pump アルゴリズムを **Algorithm 6** に示し，WorkOut アルゴリズムを **Algorithm 7** に示す．

Algorithm 6 Pump アルゴリズム:$\mathsf{Pump}(\mathbf{B}, \kappa, f, \beta, s)$

Input: n 次元格子 $\mathcal{L}$ の基底 $\mathbf{B} = \{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$, $\kappa \in \mathbb{Z}, f \in \mathbb{Z}, \beta \in \mathbb{Z}, s \in [0, 1]$
Output: 格子 $\mathcal{L}$ の近似最短ベクトルに近いベクトル

```
DB ← { }, l ← κ + β, r ← κ + β
B_[κ,r] = LLL(B_[κ,r], 0.99)
while r − l < β − f do
    v ← Sieve(B_[l−1:r])
    DB ← Babai-lift(v, B_[κ,l−1]) //[k : l] に復元したベクトルをデータベースに追加
    l ← l − 1
end while
while r − l > 0 do
    if score function により insert position i が決まったとき (κ ≤ i ≤ l) then
        DB の先頭ベクトルを position i に insert
        l ← l + 1
    else
        shrink-left(DB)
    end if
end while
```

Vol.2021-DPS-186 No.68
Vol.2021-CSEC-92 No.68
2021/3/16

Algorithm 7 WorkOut アルゴリズム:WorkOut$(\mathbf{B}, \kappa, f, f^+, \beta, s)$

Input: n 次元格子 $\mathcal{L}$ の基底 $\mathbf{B} = \{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$, $\kappa \in \mathbb{Z}, f \in \mathbb{Z}, f^+ \in \mathbb{Z}, \beta \in \mathbb{Z}, s \in [0, 1]$
Output: 格子 $\mathcal{L}$ の近似最短ベクトル

```
B' ← LLL(B, 0.99)
while L の近似最短ベクトルが見つかるまで do
    Pump(B', κ, f, β, s)
    f ← f − f+
end while
```

Algorithm 8 改良した WorkOut アルゴリズム:Improve-Workout$(\mathbf{B}, \kappa, f, f^+, \beta, s)$

Input: n 次元格子 $\mathcal{L}$ の基底 $\mathbf{B} = \{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$, $\kappa \in \mathbb{Z}, f \in \mathbb{Z}, f^+ \in \mathbb{Z}, \beta \in \mathbb{Z}, s \in [0, 1]$
Output: 格子 $\mathcal{L}$ の近似最短ベクトル or 終了した次元までの Pump された基底

```
B' ← LLL(B, 0.99)
while L の近似最短ベクトルを見つかるまで do
    Pump(B', κ, f, β, s)
    データの保存
    if 終了したい Pump 次元のとき then
        break
    end if
    f ← f − f+
end while
```

4. 改良点

本研究では，Github に公開されている G6K のプログラム [13] を元に，保存機能の追加と最適な増加値 f^+ を実験により決めた．

4.1 保存機能追加

大規模な計算機を制限なく使用できるならば，一度基底を入力することで SVP Challenge で指定されているノルム以下の近似最短ベクトルを見つけることができる．しかし，誰もが大規模な計算機を制限なしで使用できるとは限らない．G6K では基底情報や CPU time や Wall time などの解読に必要なデータを保存しながら解読を行っていないため，途中で実行が中断されるとそれまでに解読したデータが消えてしまう．そこで，G6K に保存機能を追加した．Pump アルゴリズム終了ごとに基底データを保存することにより，次の次元で計算中にプログラムが中断された場合でも，1 つ前の次元の Pump アルゴリズム後の基底データを使用して再び解読を再開させることを可能にした．Pump ごとに基底を保存する機能を追加した WorkOut のアルゴリズムを **Algorithm 8** に示す．

4.2 f^+ の値による最大 Sieve 次元の変化について

Albrecht らは論文で Pump を実行する次元を変化させる f^+ の値は 2 か 3 が最適だと示していた．そこで未解読次元に対して G6K を用いて解読するためには，どちらの値がより効率よく解読ができるのかの実験を行った．実験では，90 次元から 120 次元の各 10 次元ごとの次元で基底は SVP Challenge で生成されるものを使用した．f^+ の値は 1 から 5 の値で実験を行った．各次元の結果を図 1，図 2，図 3，図 4 に示す．

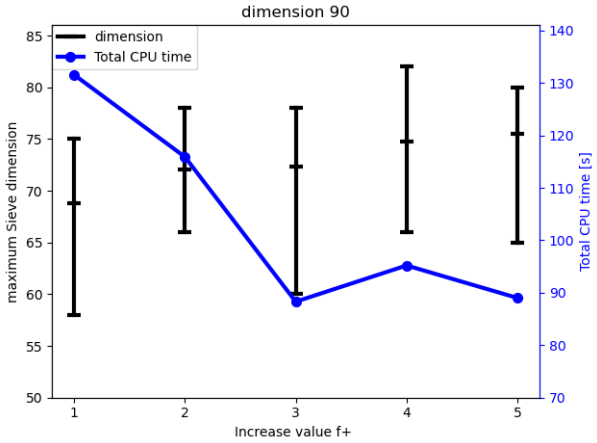

図 1 90 次元のとき

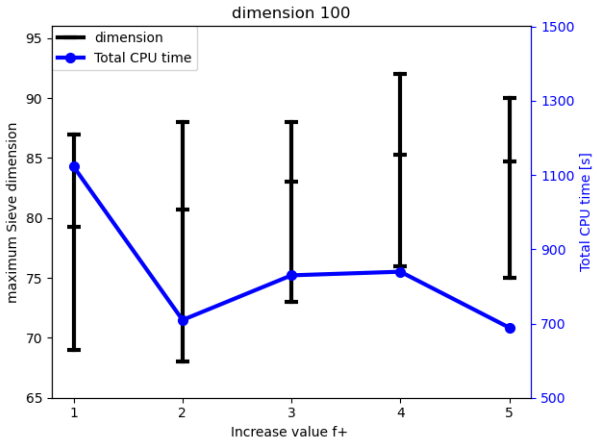

図 2 100 次元のとき

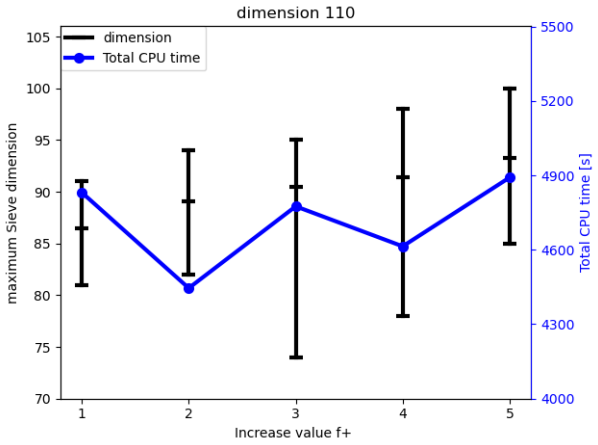

図 3 110 次元のとき

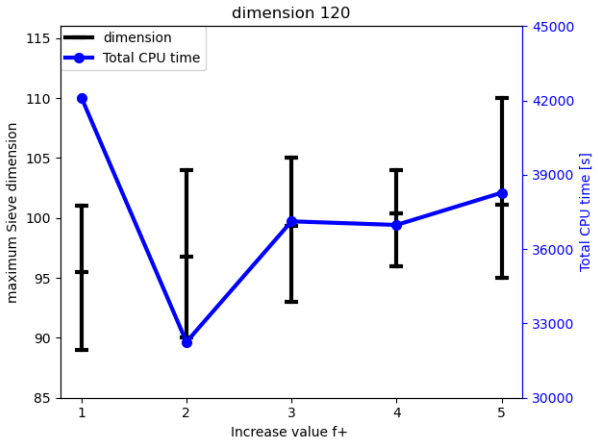

図 4 120 次元のとき

Albrecht らの論文では，f^+ の値を増加するほど解読時間が短くなるという結果だった．しかし，本実験では，解読次元が高くなるにつれ，f^+ の値 2 のときに解読時間が一番短く，f^+ に値が 3 以降だと増加する傾向が見られた．これは，f^+ の値が 3 のときより 2 のときのほうが基底簡約される回数が多く，良い基底状態での計算が多いため f^+ の値が 3 のときより 2 のときの方が，早い時間で近似最短ベクトルが見つけることができたと考える．本実験より，G6K を使用する際に設定するパラメータ f^+ の値を 2 に固定する．

5. 実験

実験は，北陸先端科学技術大学院大学 (JAIST) が所有する Large Memory PC Cluster(LMPCC) を使用した．LMPCC のシステム概要を表 1 に示す．

表 1 LMPCC のシステム概要

	Large Memory Cluster (30 nodes)
CPU	Intel Xeon G-6242 2.8GHz (16Cores x4)
Memory	1.5TB

改良した G6K と実験で求めた f^+ の値を 2 に固定し，SVP Challenge で解読されていない 154, 156, 158 次元の格子に対して解読実験を行う．格子の基底は SVP Challenge で生成される seed=0 の基底を使用した．解読結果を表 2 に載せる．

approximate factor は，各格子の Gaussian heuristic の何倍した値なのかを表している．Sieve max dim は，近似最短ベクトルが求めることに成功した際に行った Sieve の次元を表す．Wall time は，G6K のプログラムが実際に動作していた時間を示す．Total CPU time は，各 CPU が処理していた時間を合計した時間を表している．

表 2 未解読次元の解読結果

SVP dim	Norm	approximate factor	Sieve max dim	Wall time	Total CPU time
154	3200	1.02259	130	23d 4h	2623d 9h
156	3219	1.01987	130	15d 20h	1011d 16h
158	3239	1.02312	126	7d 18h	493d 16h

6. おわりに

本研究では，General Sieve Kernel(G6K) の考察と改良を行った．主に，解読に必要なデータ (基底情報や Pump を行った次元) を保存する機能の追加や改良した G6K を使用して Pump 内の Sieve の範囲の増加値を決める f^+ の値を実験により決めた．実験より，f^+ の値が 2 以降での解読時間に大きな変化は見られなかったため，メモリ使用量が少なくなる f^+ の値を 2 に固定した．最後に，改良した G6K と実験により決めた増加値 f^+ を用いて SVP Challenge の未解読次元である 154，156，158 次元の格子の近似最短ベクトルを見つけることに成功した．

謝辞 本研究は JSPS 科研費 JP20K23322 と JP19K11960 の助成を受けたものです．また，JAIST 情報基盤センターの方々には，JAIST 所有の計算機の利用などに関するサポートをいただき深く感謝いたします．

参考文献

[1] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, Vol. 41, No. 2, pp. 303–332, 1999.
[2] Post-quantum cryptography — csrc. Available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.
[3] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, Vol. 261, No. ARTICLE, pp. 515–534, 1982.
[4] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, Vol. 66, No. 1-3, pp. 181–199, 1994.
[5] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. A

sieve algorithm for the shortest lattice vector problem. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pp. 601–610, 2001.

[6] SVP Challenge. Available at `https://www.latticechallenge.org/svp-challenge/`.

[7] Martin R Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 717–746. Springer, 2019.

[8] Yuanmi Chen and Phong 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, 2011.

[9] Yoshinori Aono, Yuntao Wang, Takuya Hayashi, and Tsuyoshi Takagi. Improved progressive bkz algorithms and their precise cost estimation by sharp simulator. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 789–819. Springer, 2016.

[10] Phong Q Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, Vol. 2, No. 2, pp. 181–207, 2008.

[11] László Babai. On lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, Vol. 6, No. 1, pp. 1–13, 1986.

[12] Léo Ducas. Shortest vector from lattice sieving: a few dimensions for free. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 125–145. Springer, 2018.

[13] Github: The general sieve kernel. Available at `https://github.com/fplll/g6k`.