jh190038-NAJ

大規模並列計算による格子の最短ベクトル探索の効率化に関する研究

照屋 唯紀 (産業技術総合研究所)

概要

耐量子計算機暗号の候補の 1 つである格子暗号は，格子問題の困難性を安全性の根拠とする．最短ベクトル問題は代表的な格子問題の 1 つであり，格子の次元数が大きい程，求解が困難になると考えられている．しかし，次元数を大きくし過ぎると暗号処理の性能が著しく悪化するため，安全性と効率性を両立する適切な次元数を決定することが重要である．本課題では，大規模並列計算機を使用した最短ベクトル問題を効率的に解くアルゴリズムの開発を行う．これにより，適切な格子の次元数の決定方法の構築に貢献する．

1　共同研究に関する情報

1.1　共同研究を実施した拠点名

東京大学情報基盤センター (Reedbush-U/H/L)

1.2　共同研究分野

■超大規模数値計算系応用分野

1.3　参加研究者の役割分担

照屋唯紀 (産業技術総合研究所) (代表) アルゴリズムの調査および開発補助

柏原賢二 (東京大学) (副代表) アルゴリズムの考案，開発および実験

池上努 (産業技術総合研究所) 実装の補助, アルゴリズムの開発補助

松田源立 (成蹊大学) 確率・統計の理論的観点からの助言

2　研究の目的と意義

公開鍵暗号は情報サービスの安全性を確保するために必要な道具である．例えば，Web ブラウザなどでソーシャル・ネットワーキング・サービス (SNS) を利用する際に，第三者による通信内容の盗聴や改ざんを防ぐなど，今日の社会において広く利用されている．しかし，RSA 暗号や楕円曲線暗号など現在利用されている標準的な公開鍵暗号方式は，量子計算機を用いて理論上効率的に解読できることが知られている．そのため，量子計算機を用いて効率的に解読する方法が発見されていない耐量子計算機暗号の研究開発に注目が集まっている．

格子暗号は，耐量子計算機暗号の候補の 1 つである．格子暗号の安全性は，格子問題の困難性を根拠とする．最短ベクトル問題 (Shortest Vector Problem, SVP) は，代表的な格子問題の 1 つであり，これは格子の基底が与えられた時，その格子の最短の非ゼロ格子ベクトルを探索

する問題である．次元数が大きい場合，量子計算機を用いても求解がより困難になると考えられている．つまり，次元数を大きくすれば，格子暗号の安全性を高める事が可能である．しかし，暗号処理の性能が著しく悪化し，実用性を損なう．そのため，安全性と効率性を両立するバランスが良い次元数を決定する必要がある．このように，効率的なアルゴリズムを開発し，最高性能の計算機を使用したとしても解くことができない SVP の次元数を可能な限り精密に推定する事は，実用的な格子暗号を構築する上で重要な研究課題である．

本課題では，効率的な SVP 求解アルゴリズムの研究開発を行う．これにより，実用的な格子暗号を構築するための次元数の決定方法の確立に貢献する．特に小節 5.1 で紹介するように，最近，格子篩によって大幅な SVP 求解の効率化が達成されたことが報告されている [3]．そこで，本課題では格子篩について特に注目して研究を行なった．

3 当拠点公募型共同研究として実施した意義

先に述べたように，暗号技術の安全性の根拠となる問題の困難性は，最高性能の計算機とアルゴリズムを使用したとしても実質的に求解が不可能な程に困難でなければならない．これを実証するためには，1 つの研究室では賄う事が困難な規模の大規模な計算機実験を要する．さらに，そのような大規模な計算機に適したアルゴリズムの開発が必要不可欠である．十分な規模の計算機とアルゴリズムの開発環境を用意するために，当拠点公募型共同研究として提案し，実施した．

4 前年度までに得られた研究成果の概要

本課題は 2019 年度に採択された新規課題である．

5 今年度の研究成果の詳細

この節では，まず始めに小節 5.1 で最短ベクトル問題 (Shortest Vector Problem, SVP) とその求解に関する基礎知識，および関連研究を簡単に解説する．詳細については [1] などを参照願いたい．その後，小節 5.2 で提案アルゴリズムの概要，小節 5.3 でその実装，小節 5.4 でこの実装を実行して得た実験結果，そして小節 5.5 で実験結果に対する考察について解説する．

5.1 基礎知識および関連研究

格子 L とは，一次独立な (列) ベクトルが成す基底 $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ の整数係数線型結合が成す点 (ベクトル) 全体 $L = \{\boldsymbol{v} \mid \boldsymbol{v} = \sum_{i=1}^{n} x_i \boldsymbol{b}_i, x_i \in \mathbb{Z}^n\}$ の事を指す．ここで n は格子の次元数である．以降の議論ではフルランクの整数行列の基底で生成される格子のみを考える．

格子の基底から直交補空間を定義する．直交補空間は基底ベクトルの番号にあたる index ごとに構成でき，その index よりも前の基底ベクトル全てに直交するよう射影した空間である．つまり，直交補空間は基底に依存して定まる．本稿ではある基底 $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ に対して，index を k，格子ベクトルを $\boldsymbol{x}$ とする時，この基底を用いた index k の直交補空間は $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{k-1}$ 全てに直交する空間であり，この空間に射影した格子ベクトルを $\pi_k(\boldsymbol{x})$，射影格子を $\pi_k(L)$ と書く．本稿では射影された格子ベクトルのことも格子ベクトルと呼ぶ．

SVP とは，格子を生成する基底を入力に取り，非ゼロで長さが最短の格子ベクトルを探索

する問題である．格子の次元数に対して，求解にかかる計算時間のオーダーが指数関数的に爆発する．

格子の最短ベクトルの長さは，ガウスの経験則 (Gaussian heuristic) によって推定できる．ガウスの経験則によって推定した格子 L の最短ベクトルの長さを $\mathrm{GH}(L)$ と表記する．また，直交補空間に対してもガウスの経験則を考えることができ [3]，これを $\mathrm{GH}(\pi_k(L))$ と表記する．

SVP Challenge [2] は，SVP の生成器，つまり格子を生成する基底を公開している Web サイトである．各格子 L について，$1.05 \cdot \mathrm{GH}(L)$ 未満の長さとなる格子ベクトルの投稿を受け付けており，その次元数や短さなどを元にランキングを行っている．SVP は，厳密に最短の格子ベクトルを解とするだけでなく，SVP Challenge のように問題の定義方法がいくつか存在する．本課題では SVP Challenge と同様に，$1.05 \cdot \mathrm{GH}(L)$ 未満の長さの格子ベクトルを探索する問題を考える．

SVP の求解アルゴリズムには，主に格子基底簡約，格子点列挙 (Enumeration, ENUM．または格子点サンプリング)，格子篩 (sieving) の 3 種類がある．格子点列挙と格子篩は，基底を入力に取り，短い格子ベクトルを探索する．格子基底簡約の代表的なアルゴリズムは，LLL アルゴリズムや BKZ アルゴリズムである．本稿では，格子基底簡約アルゴリズムは，基底とさらに補助的にいくつかの格子ベクトルを入力に取り，入力された基底に対して簡約処理を行なった基底を出力するアルゴリズムであるとする．基底に簡約処理を行うと，経験的に良い基底に変換される．ここで良い基底とは，直観的には基底ベクトルが互いに直交に近いことを指す．そして良い基底を格子点列挙と格子篩に入力することで，経験的にその性能が向上する．これら性質を利用して，それぞれのアルゴリズムを適切に組み合わせる事が効率的な解法を構成する上で重要である．

現在最も高速な解法は，射影による次元削減を行った射影格子に対して格子篩を実行し，次に Babai の最近平面アルゴリズム (Babai lift) を実行することで，射影された格子ベクトルを元の空間の格子ベクトルに変換して短い格子ベクトルを探索する，射影格子篩と呼ばれる方法と格子基底簡約を組み合わせた方法である．なお，Babai lift は，直観的には格子点列挙 (および格子点サンプリング) を原点に近い格子ベクトルを出力するように限定して実行した場合に一致する．この解法の実装の 1 つである G6K [3] は SVP Challenge で出題されている 155 次元という非常に高い次元数の近似 SVP の求解に成功している．この記録は 4 個の Xeon E7-8860 v4 (合計 72 コア) と 512GiB RAM を搭載する計算機を用いて達成された．なお，2020 年 5 月 13 日時点において求解が報告されている最大の次元数は 170 次元である．SVP Challenge のランキングに登録されている optional notes に記述されている簡易的な解説によると，これは G6K から派生した GPU を利用した実装による求解結果であると報告されている．

5.2 提案アルゴリズムの概要

ここで，本研究において提案するスーパーコンピュータ上で分散並列計算できるように実装した格子篩のアルゴリズムを解説する．

提案アルゴリズムは，射影による次元削減で順次生成された次元の異なる複数の直交補空間ごとに短い格子ベクトルのリストを保持し，それぞれの直交補空間で射影格子篩の処理を行いながら，他の直交補空間の短い格子ベクトルも探索する．ここで，格子篩の処理とは，大量の 2 つの格子ベクトルの組みの和・差を計算す

ることで，短い格子ベクトルを探索することである．他の直交補空間の短い格子ベクトルを探索する時，高次元の直交補空間の短い格子ベクトルの探索には Babai lift を，低次元の直交補空間の短い格子ベクトルの探索には射影を用いる．これら探索処理により，複数のベクトルリストを組み合わせながら全体を徐々に短い格子ベクトルに置き換えていくことで，短い格子ベクトルを探索する．

提案アルゴリズムにおける基本的な停止条件の概要について述べる．リストから取り出した格子ベクトルの組みの和・差を計算したとしても，リストに保存すべき短い格子ベクトルが得られない状況を既約とし，停止する．保存すべき格子ベクトルの選択については，その長さに対して閾値を設定するなど，長さを基準として判断する．

上で述べたように，基本的に得られた格子ベクトルの長さを基準としてリストに保存するか否かを判断するが，効率化のためには重複した計算を排除する必要がある．既に得られた格子ベクトルと同一の格子ベクトルの保存を避けることで，重複を排除する．格子ベクトルの整数座標に対して，0 からある値までの自然数値をとるハッシュ関数を用意し，同一ハッシュ値ごとに直交補空間それぞれにおいて最短の格子ベクトルのみを保持する．但し，格子ベクトル $\boldsymbol{v}$ と $-\boldsymbol{v}$ のハッシュ値が等しくなるようなハッシュ関数を用いる．これにより，格子ベクトルの重複が容易に判定できる．

このように，格子ベクトルの長さは直交補空間に射影した射影座標系における長さを考えるが，格子ベクトルのハッシュ値は元の格子の空間における座標系の整数値を用いる．

提案アルゴリズムは格子ベクトルのリストに対して格子篩の計算を何度も繰り返し行うが，この繰り返し 1 回の処理にラウンド番号を割り当て，重複した計算を避けるために，格子ベクトルを得た際のラウンド番号をタグとしてその格子ベクトルに付与した．

アルゴリズムの大規模並列化にあたり，格子ベクトルのリストを分割することで，単一ノードのメモリ量の制限を超えて大量のベクトルを保持できるように実装した．

5.3 提案アルゴリズムの実装

Algorithm 1 に提案アルゴリズムの実装の概要を示す．

並列化に関しては, MPI を用いた SPMD (Single Program Multiple Data) モデルに基づいて実装し，MPI rank によって割り当てられるプロセス番号に応じて，読み込む格子ベクトルを変化させている．F は分割リストにつけた番号を要素とする集合である．格子ベクトルのリストは，ハッシュ値に応じた場所に分散させて保存する．

アルゴリズムを実行する際には，入力としてある程度簡約 (例えば，LLL や BKZ などで簡約) した基底 B を与え, 計算の間は変更しない．また，別途計算 (例えば，格子点列挙などで計算) した初期格子ベクトルも与えられているとする．初期格子ベクトルのタグはすべて 0 とする．

Index i の直交補空間に対する格子ベクトルのリストを S_i と書く．それぞれの S_i に対して，タグとして付与するラウンド番号 r を持つ格子ベクトルの集合を $S_{i,r}$ と書く．さらに，$S_{i,r}$ のうち，m 番目の分割された部分リストに保存されている格子ベクトルの集合を $S_{i,r,m}$ と書く．また，I_s は最大の直交補空間の index $i_{\max}$ から最小の index $i_{\min}$ までを並べた有限列である (大きい順に並んでいる)．読み込んだ格子ベクトル集合のペアごとに格子ベクトルの和と差を計算し，そしてその結果に Babai lift や射影 π_i を適用することで，短い格子ベクトル

Algorithm 1 提案格子篩アルゴリズム

Require: 基底 B，大きい順に並んだ直交補空間の index の有限列 $I_s = (i_1, i_2, \ldots, i_N)$ ($i_1 = i_{\max}$ および $i_N = i_{\min}$ である)，各直交補空間で短いと判断するための自乗長さの有限列 $T = (T_{i_1}, T_{i_2}, \ldots, T_{i_N})$，初期格子ベクトル集合 $S_{i,0}$ ($i \in I_s$)，プロセスの数 M，プロセス番号 (MPI rank) $m \in \{0, 1, \ldots, M-1\}$.

Ensure: 全ての格子ベクトルの集合 $S = \bigcup S_{i,r}$.

1: $r = 1$ とする.
2: **loop**
3: **for all** $k = 1$ **to** N **do**
4: **for all** $\ell = 0$ **to** $r - 1$ **do**
5: **if** $\ell = r - 1$ **then**
6: $F = \{m, (m+1) \bmod M, \ldots, (m + \lfloor M/2 \rfloor) \bmod M\}$ とする.
7: **else**
8: $F = \{0, 1, \ldots, M-1\}$ とする.
9: **end if**
10: **for all** $f \in F$ **do**
11: **for all** $(\boldsymbol{v}_1, \boldsymbol{v}_2) \in S_{i_k, r-1, m} \times S_{i_k, \ell, f}$ **do**
12: 和 $\boldsymbol{v}_1 + \boldsymbol{v}_2$ と差 $\boldsymbol{v}_1 - \boldsymbol{v}_2$ を計算し，index i の直交補空間で短い方を $\boldsymbol{v}$ とする.
13: $\boldsymbol{v}$ が index i_k の直交補空間において $|\boldsymbol{v}|^2 < T_{i_k}$ であり，さらにハッシュ値が $\boldsymbol{v}$ と同じ S_{i_k} に含まれる全てのベクトルよりも短いならば，タグ r を付与して $S_{i_k, r}$ に保存.
14: $\boldsymbol{v}$ を index j ($j > i_k$ かつ $j \le i_{\max}$) の直交補空間に射影して $\boldsymbol{v}'$ とする時，$|\boldsymbol{v}'|^2 < T_j$ であり，さらにハッシュ値が $\boldsymbol{v}$ と同じ S_j に含まれる全てのベクトルよりも短いならば，タグ r を付与して $S_{j,r}$ に保存.
15: $\boldsymbol{v}$ を index j ($j < i_k$ かつ $j \ge i_{\min}$) の直交補空間のベクトルに Babai lift により変換して $\boldsymbol{v}'$ とする時，$|\boldsymbol{v}'|^2 < T_j$ であり，さらにハッシュ値が $\boldsymbol{v}$ と同じ S_j に含まれる全てのベクトルよりも短いならば，タグ $r-1$ を付与して $S_{j,r-1}$ に保存.
16: **end for**
17: **end for**
18: **end for**
19: **end for**
20: 新規に保存したベクトルが 1 個も無かった，または目的のベクトルが得られたら終了.
21: r をインクリメントする.
22: **end loop**

を探索する.

$I_s = (i_1, i_2, \ldots, i_N)$ とした時，T を短いと判断するための自乗長さの有限列 $T = (T_{i_1}, T_{i_2}, \ldots, T_{i_N})$ とする．Index i_k の直交補空間においては，Babai lift または射影によりこの空間の格子ベクトルに変換し，その自乗長さが T_{i_k} 未満なら短いと判断する.

格子篩の処理において，index i の直交補空間において格子ベクトルの組みの和・差を計算している時, i より小さい index で短い格子ベクトルを発見した時には, (現在のラウンド番号 − 1) のタグ番号を格子ベクトルに付与して保存し，i より大きい index で短い格子ベクトルを発見した時には，現在のラウンド番号をタグ番号として格子ベクトルに付与して保存する.

本課題で実装したプログラムでは，Algorithm 1 のループの内側の 11 行目から 16 行目の処理の概要は，次のようになっている.

1. 分割されたリストから複数の格子ベクトルをまとめて読み込み，その射影座標を計算する．
2. GPU 上で (後で解説するように cuBLAS を使用して)，これら格子ベクトルの組の射影座標の和と差の長さを計算し，その中から T を参照して index i の直交補空間において短い格子ベクトルに絞り込む．また，絞り込まれた和と差の格子ベクトルに対して，GPU 上で Babai lift と射影の計算を行い，得られた格子ベクトルの長さがあらかじめ T により各 index ごとに定めた長さよりも短いか調べる．短いならば，格子ベクトルに対応する格子ベクトルの組のリストを出力．
3. 前段階で絞り込まれた格子ベクトルの組に対して，CPU 上で改めて和または差の計算および Babai lift の計算を行なって格子ベクトルの整数座標値を求め，そしてハッシュ値を求める．次に，直交補空間ごとに同じハッシュ値の全ての格子ベクトルよりも短いか調べる．短いならば，これを保存する．

格子ベクトルの和と差の計算は射影座標系において行い，直交補空間における長さで短さを判断する．なお，$\boldsymbol{x}, \boldsymbol{y} \in L$ に対して，index i の直交補空間の和と差について $\pi_i(\boldsymbol{x}) + \pi_i(\boldsymbol{y}) = \pi_i(\boldsymbol{x} + \boldsymbol{y})$ である．実装上は，射影座標系における座標の値を浮動小数点数で表現して計算する．具体的には $|\boldsymbol{x} \pm \boldsymbol{y}|^2 = |\boldsymbol{x}|^2 + |\boldsymbol{y}|^2 \pm 2\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ を計算する．GPU 上で計算する格子ベクトルの組の候補の絞り込みも，射影座標系を用いる．絞り込み後，CPU 上で元の整数値による座標に戻す処理を実施することで，丸め誤差の蓄積を避けている．

大規模計算機環境では，計算ノードごとにメモリが独立しており，すべてのノードで共通に高速にアクセスできるようなメモリが使用できない場合が多い．我々は，ノード間の MPI を用いたプロセス並列により，CPU 間でメモリが共有できない分散メモリ環境において有効なアルゴリズムを実装することを目指した．特にベクトルリストの分割保持に MPI-IO を用いて，格子篩に用いる各直交補空間ごとに短いベクトルの情報をファイルに保存する．MPI-IO を使うことで，異なるプロセスからの出力ファイルを単一のファイルにまとめることができ，また，必要な格子ベクトルの情報を好きな場所に読み書きできる．そして格子ベクトルの保存位置を分割することにより，効率的な並列化を実現することができたと考えられる．

Algorithm 1 の 11 行目から始まる処理において，2 つの格子ベクトルの集合から格子ベクトルを読み込みその和と差を計算する際には cuBLAS を使用した．cuBLAS で処理するためにそれぞれの格子ベクトルの集合を約 2000 個のブロックに分割した．2000×2000 程度の格子ベクトルの組み $\boldsymbol{x}, \boldsymbol{y}$ において $\boldsymbol{x} + \boldsymbol{y}$ の長さの計算は $|\boldsymbol{x}+\boldsymbol{y}|^2 = |\boldsymbol{x}|^2+|\boldsymbol{y}|^2+2\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ となるので，この計算を次のようにまとめて cuBLAS を用いた行列計算に帰着させた．2 つの格子ベクトルの集合 S_A, S_B は，それぞれ m_1 個，m_2 個の格子ベクトルが含まれているものとする．それぞれの集合に含まれる格子ベクトルの射影座標系の値を列ベクトルとして並べて作成した行列をそれぞれ $A = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{m_1})$，$B = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{m_2})$ とする．行列積 ${}^tAB$ を計算することにより $m_1 \times m_2$ 行列を作成する．行ベクトル X の i 番目の要素に $|\boldsymbol{x}_i|^2$ を，同様に行ベクトル Y の i 番目の要素に $|\boldsymbol{y}_i|^2$ を保存する．次に m_2 次元の定数行ベクトル $U = (1, \ldots, 1)$ を用いて ${}^tXU$ を計算し $m_1 \times m_2$ 行列を作成する．同様に m_1 次元の定数行ベクトル $V = (1, \ldots, 1)$ を

用いて ${}^tVY$ を計算し $m_1 \times m_2$ 行列を作成する．これら 3 つの $m_1 \times m_2$ 行列の和を取ることで，$i \in \{1, \ldots, m_1\}$ および $j \in \{1, \ldots, m_2\}$ について $|\boldsymbol{x}_i + \boldsymbol{y}_j|^2 = |\boldsymbol{x}_i|^2 + |\boldsymbol{y}_j|^2 + 2\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$ をまとめて計算させた．$\boldsymbol{x} - \boldsymbol{y}$ の長さの計算も同様にして行う．具体的な API としては `cublasSger` と `cublasSgemm` を用いた．

5.4 提案アルゴリズムの実験結果

ここでは，提案アルゴリズムを実装したプログラムを，東京大学の大規模並列計算機システム Reedbush-H 上で実行した結果について解説する．Reedbush-H は，各ノードに Xeon E5-2695v4 の CPU を 2 個ずつ搭載し，GPU は NVIDIA Tesla P100 を各ノードごとに 2 個づつ搭載している．

基本的に 4 ノードを使用し，入力は 154 次元の格子 L の基底，$i_{\min} = 20$，$i_{\max} = 60$ とした．効率的な計算のために，何度か予備実験を行い，その結果を元に保存する基準の長さ T を設定した．具体的には，保存すべき格子ベクトルが増加し過ぎて計算時間に影響が出ないように，ラウンド 1 と 2 の計算ではすべての index i において $\mathrm{GH}(\pi_i(L))$ の 2 倍以下としたが，ラウンド 3 では基準を 1.8 倍以下と厳しくした．ラウンドが進むことで，T で指定する保存する基準の長さ以下であるベクトルの個数がどのように増加するのかを，表 1 にまとめた．長さが基準以上になった格子ベクトルは，リストから削除した．その影響で新規の格子ベクトル登録数がやや減少している．

また，格子篩を行う直交補空間を表す index $i = 46$ において，Babai lift や射影を用いて得られる格子ベクトルについて，index ごとの短い格子ベクトルの個数のグラフを図 1 に示す．これは，保存した格子ベクトルの個数ではなく，T で指定される長さ未満の長さとなった格子ベクトルの個数を数えたものである．

表 1　ラウンドごとの格子ベクトルの個数

		ラウンド		
Index i	開始時	$r = 1$	$r = 2$	$r = 3$
60	1	10482	12574	9026
50	14	19861	23948	13160
40	69	25300	27303	11685
30	90	23190	23898	6827
20	0	22103	22447	2185

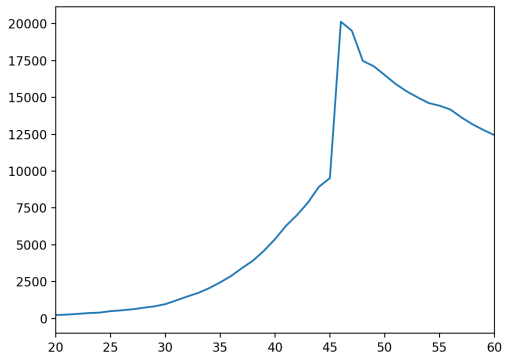

図 1　Index $i = 46$ の格子篩計算において，Babai lift や射影により生成した各 index j (横軸) の直交補空間における格子ベクトルについて，その長さが T で指定する長さ未満となった格子ベクトルの個数 (縦軸)

5.5 考察

SVP Challenge で出題されている問題の求解および記録の更新を目的にアルゴリズムを設計しプログラムを作成したが，現在のところ高次元での記録更新は達成できておらず，また性能は十分に効率的とは言えない．また，実験の際には各直交補空間において十分に短い格子ベクトルを発見する前に，早い段階で既約となり停止したことも観測された．このことも，提案アルゴリズムは効率的ではないことを示している．その原因としては，次が考えられる．

1. 格子篩の処理の対象となる直交補空間が限

定されていることが考えられる．

2. 提案アルゴリズムは入力した基底を固定しており簡約していないことが考えられる．より良い基底を入力に取って射影格子篩を実行すると，経験的に性能が良くなることが知られている．そのため，何らかのタイミングで格子基底簡約を行うべきだろう．

また，得られた格子ベクトルを保存すべきか否かを判断する T は予備実験により定めたが，予備実験を行わずに定める方法が必要であると考えられる．

このように，提案アルゴリズムは問題点や改善すべき点が残されている．

6　今年度の進捗状況と今後の展望

提案アルゴリズムは小節 5.5 で述べたように改善すべき点が残されている．これらを分析しながら今後も研究を進め，より効率的なアルゴリズムを構成し，その高速な実装を行うことが今後の課題である．

■参考文献

[1] 青野良範，安田雅哉，“格子暗号解読のための数学的基礎”，近代科学社

[2] “SVP Challenge”, 2010, https://www.latticechallenge.org/svp-challenge/

[3] Albrecht et al., “The General Sieve Kernel and New Records in Lattice Reduction”, EUROCRYPT 2018, https://github.com/fplll/g6k

7　研究業績一覧（発表予定も含む）

学術論文 （査読あり）

なし．

国際会議プロシーディングス　（査読あり）

なし．

国際会議発表　（査読なし）

なし．

国内会議発表　（査読なし）

なし．

その他（特許，プレス発表，著書等）

なし．