完全準同型暗号における bootstrap problem 及び relinearize problem の厳密解法の高速化

佐藤　宏樹† 　石巻　優† 　山名　早人††

† 早稲田大学 基幹理工学研究科　〒 114–1919 東京都新宿区大久保 3-4-1
†† 早稲田大学 理工学術院　〒 114–1919 東京都新宿区大久保 3-4-1
E-mail: †{hsato,yuishi}@yama.info.waseda.ac.jp, ††yamana@waseda.jp

あらまし　完全準同型暗号 (FHE) は秘密鍵を用いることなく任意の演算が暗号文上で可能な暗号方式である．FHE は，データの秘密を保持したまま解析や検索を行う技術への応用が期待される．FHE 上での演算は平文上での演算と特徴が異なるため，FHE に適したアルゴリズムや最適化が必要である．特に，暗号化状態での計算を継続的に行うためには bootstrapping や relinearization と呼ばれる処理を暗号文に施す必要がある．bootstrapping や relinearization は FHE の中で計算量が大きいため，計算のデータフローグラフ上でのこれらの処理の実行スケジューリングは，FHE による計算全体の処理時間に影響を与える．それぞれのスケジューリング問題は bootstrap problem，relinearize problem と呼ばれ，ともに NP 困難であり，制約式を列挙すると整数線形計画問題 (ILP) となる．変数と制約式の個数はデータフローグラフの辺と頂点の個数に比例するため，最適化対象のグラフが大きくなると厳密解を得ることが困難である．本稿では，出力される ILP の解は厳密解から変えずに，ILP の変数と制約式を削減し，本処理を適用しない場合よりも高速に解を求める手法を提案する．評価実験では，多くの場合で提案手法により頂点数を約 3〜4 割削減し，厳密解を求める最適化の計算時間も約 3〜4 割削減した．

キーワード　完全準同型暗号，bootstrap，relinearize，最適化

1 はじめに

近年，国や企業などの組織が保有するデータ量が爆発的に増加するにつれ，保有するデータを解析し活用するために計算資源としてクラウドを利用するケースが増加している．しかし，医薬品や遺伝子情報などの機密性の高いデータに対する各種処理をクラウドという第三者のサーバ上で行う場合，サーバからの機密情報漏洩が懸念される．

データの秘密を保持したまま解析を行う秘密計算技術のうち，本稿では完全準同型暗号 (FHE, Fully Homomorphic Encryption) [1] を扱う．FHE は，秘密鍵を用いることなく任意の演算を暗号文上で行うことのできる暗号方式である．FHE を用いることでデータの秘密を保持したままデータの解析や検索が可能となり，クラウドコンピューティングへの応用が期待される．

ある演算を FHE 上で行うとき，その計算回路におけるデータ依存関係をデータフローグラフにより表現することができる．暗号化したまま，すなわち準同型に演算したい計算を，データフローグラフにより表現することで，既存のコンパイラと同様に，データフローグラフ上で演算を最適化できる．しかし FHE での演算には，暗号文の持つ平文に対する演算処理だけではなく，これから述べるような暗号文のメンテナンスのためのみに用いられる演算処理がある．これらの処理は暗号文の保持する平文は変更しないためデータフローグラフ上で明示的に示されず，既存のコンパイラにおける最適化技術を直接適用できない．

暗号文メンテナンスのための処理の一つが bootstrapping である．そもそも FHE は格子に基づく暗号であり [2]，暗号文にノイズ成分が含まれる．暗号文中のノイズは準同型演算を行うたびに増加し，ノイズが一定量以内でないと正しく復号できないため，1 つの暗号文に対して連続で行える演算回数に限界がある．この計算回数の限界に対し，Gentry により提案された bootstrapping [1] と呼ばれる手法を用いると，暗号化状態を維持したままノイズを削減することが可能になり，任意の回数の計算できるようになる．しかし，bootstrapping は時間・空間計算量がともに膨大であり，1 回あたり数秒〜数分を要するため，FHE を用いた計算のボトルネックとなる．暗号文を正しく復号できる状態を維持したまま，データフローグラフ上のどの位置で bootstrapping を行うかにより，必要となる bootstrapping の回数が変化する．この bootstrapping の回数を最小化する最適化問題は bootstrap problem と呼ばれる [3].

もう一つの暗号文メンテナンスのための処理が，relinearization である．Ring Learing-With-Errors (Ring-LWE) 問題 [4][1] を基に構成される FHE では，暗号化直後の暗号文は長さ 2 の多項式のベクトルといえる．暗号文上で準同型に乗算をするにつれて，ベクトルの長さ（以下，次元）は大きくなる．さらに，暗号文の次元が大きくなると，比例して準同型演算の計算量も大きくなる．そこで FHE では，relinearization と呼ばれる処理を行うことで，暗号文の次元を削減することができ

1：効率的に解くことが困難とされている問題の一つ．

表 1　本稿で用いる主な記号

記号	意味
L	FHE の暗号化直後の暗号文のレベル
N	FHE の bootstrapping 直後のレベル
l	FHE の暗号文のレベル
$len(c)$	FHE の暗号文 c の次元
G	有向グラフ $G = (V, E)$
V	グラフにおける頂点集合
E	グラフにおける有向辺集合

表 2　HElib [10] による計算時間例

処理	計算時間（秒）
暗号化	0.158
復号化	0.044
準同型加算	0.004
準同型乗算	0.170
Bootstrapping	333.827

Core i7-1479@3.6GHz (8thread) 上で実験し，FHE のパラメータは $p = 2, m = 27311, L = 26\,(\lambda = 110.1)$ とした．準同型乗算中には relinearization の時間も含む．

る．しかし，relinearization 自体にも準同型乗算と同程度の計算量を必要とする．Bootstrapping を除くと，FHE の演算の中で relinearization と準同型乗算が計算量が大きい．データフローグラフ上で，計算の順序やデータの依存関係は変更せずに，relinearization を実行するタイミングや回数を決定して，回路全体での計算コストを最小化する最適化問題は relinearize problem と呼ばれる [5].

Bootstrap problem, relinearize problem はともに，NP 困難であることが知られている [3], [5]．既存研究では，近似アルゴリズム [6] やデータフローグラフがある特別な条件を満たす場合の多項式時間の解法 [5] が提案されている．しかし，bootstrap problem や relinearize problem の厳密解を求めるには，制約条件と目的関数を定式化して整数線形計画問題を解く必要がある．ソルバを用いることで厳密解を得ることができる [7] が，必要な時間・空間計算量はグラフの大きさに指数的に増加する．そこで本稿では，準同型加算ではノイズ増加が小さく，暗号文の次元が増えないことに注目し，データフローグラフに対し前処理を施し，問題を解く際に考慮すべき頂点数を削減する．これにより整数線形計画問題の変数と制約式の個数を削減し，本処理を適用しない場合よりも高速に厳密解を求める手法を提案する．

本稿は以下，**2.** で FHE と bootstrap problem と relinearize problem の既存研究を説明する．その後 **3.** で提案手法を説明し，**4.** で実験と提案手法の評価検証を行う．最後に **5.** でまとめを述べる．

2　研究背景と既存研究

本稿で用いる記法を表 1 に示す.

完全準同型暗号（FHE）とは，暗号化状態で平文に対して加算を行える加法準同型性と乗算を行える乗法準同型性の両方の性質を備える公開鍵暗号である．1978 年に Rivest らが初めて FHE の概念を示し [8]，2009 年に Gentry が具体的な構成法を発表 [1] した．以降，FHE に関連した研究が継続的に行われている．

2.1　Bootstrap Problem

FHE の暗号文はノイズを含んでおり，ノイズの大きさは暗号化直後に最小で，準同型加算や準同型乗算を繰り返すたびに増加する．ノイズの量がある閾値を超えると暗号文は正しく復号できなくなるため，bootstrapping と呼ばれる処理を実行することで，平文の値はそのまま，暗号文に蓄積したノイズを削減することができる．しかし，表 2 に示すように，準同型加算や準同型乗算に対し，bootstrapping には数百〜数千倍の時間を要する．したがって，bootstrapping を含む FHE の計算回路に演算に要する時間は bootstrapping の回数にほぼ比例する．

また，本稿では FHE のうち，leveled FHE の使用を前提とする．Leveled FHE は，bootstrapping を用いずに演算可能な乗算回数の上限を，鍵生成時に渡す「レベル」パラメータ L により指定できる FHE である．Leveled FHE を実現する FHE の一つに BGV 方式 [9] がある．BGV 方式は，準同型演算中のノイズ増加を制御する技術の一つである modulus switching を備えている．具体的に BGV 方式では，公開パラメータに L 個の奇数 $q_L > q_{L-1} > \ldots > q_1$ を含む．暗号文空間の多項式の係数の法にはこれらの値が用いられ，特に法が q_l の暗号文をレベル l の暗号文と呼ぶ $(1 \leqq l \leqq L)$[2]．暗号化直後の暗号文はレベル L であり，準同型演算を行った後に modulus switching を行い適宜レベルを下げることで，もともとは準同型演算回数に対して指数的に増加したノイズを，線形程度の増加に抑制できる．必要な bootstrapping の回数を減らすために L を大きくとると，保証されるセキュリティレベルが減少するとともに，準同型演算と bootstrapping の計算時間は増加する．すなわち，レベルを大きくするだけでは，計算の高速化は行えず，セキュリティレベルも低下する．bootstrapping の必要回数を減らすためには L を大きくするだけでは不十分である [3]．

Leveled FHE のレベルパラメタ L と計算回路のデータフローグラフが与えられたとき，正しく復号できる状態を維持したまま，グラフ中における bootstrapping の実行回数を最小化する最適化問題を bootstrap problem と呼ぶ [6], [7]．本稿では特に，modulus switching [9] を用いた leveled FHE による場合について述べる．

完全準同型暗号において，準同型加算によるノイズの増加量は準同型乗算によるノイズ増加量と比較して小さい．そこで，bootstrap problem では，modulus switching を用いた leveled FHE について「準同型加算の際のノイズ増加は無視し，ノイズが大きく増加する準同型乗算の際，その計算直後に暗号文のレベルを 1 つ下げる」とモデル化する [3], [6], [7]．具体的には，

2：既存研究ではレベルの表現方法が 0 や 1 から始まり増加するものもあるが，本稿では FHE の実装の一つである HElib と同一となる表現を選択した．

3：bootstrap problem の既存研究では，簡単のため L を固定した状態で最適化問題を議論している．

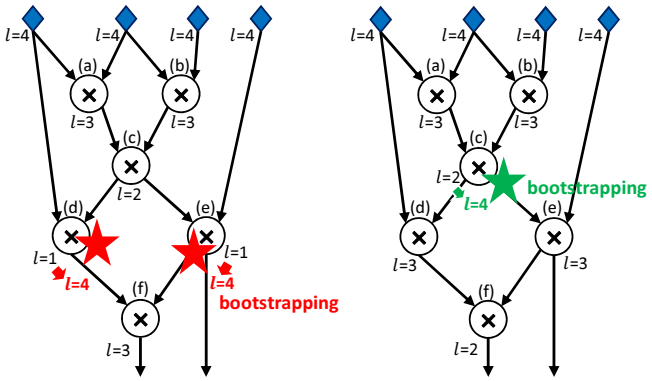

図 1　Bootstrap Problem の例

レベル l_1 とレベル l_2 の暗号文を入力としたとき，準同型加算ではレベル $\min(l_1, l_2)$ の暗号文を出力し，準同型乗算ではレベル $\min(l_1, l_2) - 1$ の暗号文を出力するとモデル化する．また，bootstrapping 直後の暗号文のレベルを $N(\leqq L)$ とする．

計算回路のデータフローグラフ $G = (V, E)$ を，入力数が 0〜2 の頂点の集合 V と頂点間をつなぐ有向辺の集合 E により定義する．計算回路のゲートが頂点に，ワイヤが有向辺に対応する．

例として，図 1 にデータフローグラフを示す．ここで，$L = N = 4$ と設定する．回路の入力が図の上部の菱形の頂点であり，上部から下部に向けて計算を行う．それぞれの頂点の横に記載された $l = x$ で示される x の値が，その頂点での計算直後のレベルを表す．ナイーブな手法として「頂点での計算直後のレベルが 1 に到達したら bootstrapping を行う」場合を考える（図 1 左）．このとき，bootstrapping は `(d)` と `(e)` の計算直後で行うこととなり，2 回必要となる．これに対し，1 つ手前の頂点 `(c)` の計算後に bootstrapping を行う場合（図 1 右），回路中で bootstrapping を行う回数は 1 回となり，回路全体での計算時間を削減できる．

Bootstrap problem は，Lepoint と Paillier が 2013 年に初めて指摘した [3]．2015 年に Paindavoine と Vialla がグラフ理論を用いた bootstrap problem に関する研究成果を発表し，整数線形計画問題 (ILP, Integer Linear Problem) に本問題を変換し，厳密解を求める手法を示した [7]．また，2017 年に Benehamouda らは，$L = N$ を満たす場合における bootstrap problem の多項式時間の近似解法を示した [6]．筆者らは 2018 年，計算回路が繰り返し処理を含む場合の最適化手法を提案した [11]．本稿では，Paindavoine と Vialla の ILP による厳密解の求まる解法をナイーブな解法とし，高速化の対象とする．

2.2　Relinearize Problem

Ring-LWE 問題を基に構成される FHE では，暗号化直後の暗号文は長さ 2 の多項式のベクトルで表される．暗号文上で準同型乗算を行うにつれて，ベクトル長（以下，次元）は大きくなる．ここで，準同型加算や準同型乗算を計算する際の計算時間は出力暗号文の次元に比例する．したがって，暗号文 c の次元を $len(c)$ と表記したとき，比例定数 k_a, k_m を用いて，準同型加算の計算時間は $k_a \max(len(c_1), len(c_2))$，準同型乗算の計算時間は $k_m(len(c_1) + len(c_2) - 1)$ と表現できる．なお，relinearize problem において k_a と k_m は，他の FHE のパラメータを決定した後に実測により得る値とする．

準同型の乗算を継続すると暗号文の次元は増加する一方であるが，relinearization と呼ばれる処理を行うことで暗号文の次元を削減することができる．具体的には，次元 $len(c)$ の暗号文を relinearization により，平文の値を変えることなく次元 $len(c) - x$ の暗号文に変換することができる．Relinearization に要する時間は，減少させる次元数に比例し，パラメータ k_r を用いて $k_r x$ と表現でき，準同型乗算 1 回と同等程度の計算コストを要する．なお，暗号文の次元は 1 にすることはできないため 2 が最小となる．また，準同型乗算と比較して準同型加算は軽いため，以降では準同型加算に要する計算時間は無視して議論を進める．

既存の FHE を用いた実装やアプリケーションの研究では，簡単のため，すべての乗算処理の後に relinearization を行うことで暗号文の次元を常に 2 に維持している場合が多い．しかし，relinearization と乗算に要する時間の変化は，パラメータ k_m, k_r と計算回路に依存して最適な選択は異なるため複雑である．計算回路と FHE のパラメータ k_m, k_r が与えられたとき，データフローグラフには変更を加えず，relinearization を実行する頂点とその回数を決定して，回路全体での計算コストを最小化する問題は relinearize problem と呼ばれる [5]．

Relinearize problem は bootstrap problem と同様に暗号文の次元に関する制約式を列挙し，目的関数として回路全体の計算時間を立式することで，整数線形計画問題に変換することができる．

3　提 案 手 法

Bootstrap problem と relinearize problem は共通して，対象とするデータフローグラフが大きくなると，ILP の変数と制約式の個数が大きくなり，ILP の解を求めるのに要する計算時間が増大するという問題点がある．具体的には，データフローグラフ $G = (V, E)$ から ILP への変換により，立式する変数の個数と制約式の個数はともに $\mathcal{O}(|V|)$ である．ILP の計算時間は，制約式と変数の個数に対して指数的に増加する．グラフの頂点数削減は変数と制約式の個数の削減につながり，ILP の計算時間を削減できる．

そこで本稿では，ILP の計算量オーダーは変わらないものの，そのまま制約式を立式した場合と比べて制約式の個数を削減可能な手法を提案する．なお，簡単のため，以下は bootstrap problem に適用する場合のみを説明する．Relinearize problem に対しても，bootstrapping を relinearization と読み替え，レベルの変化を暗号文の次元の変化と読み替えることで同様に説明が可能である．

本稿では準同型加算時のノイズを無視していることを利用し，ILP に変換する前に，bootstrapping の配置決定において考慮する必要のない頂点をデータフローグラフから削除するこ

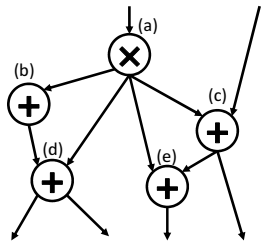

図 2　乗算直後に加算 gather 複数ある回路の例

とで，データフローグラフを小さくすることを目指す．例として，図 2 にデータフローグラフの一部を示す．図中では，(a) の頂点のみ乗算を表しており，その他の頂点は準同型加算を表す．また，(c) には図示範囲外からの入力がある．

ここで例えば ILP の結果として (b) 又は (d) の頂点で bootstrapping を行う状況を得たときを考える．3 頂点 (b)，(d)，(a) の間には暗号文のレベルが変化しうる頂点がないため，(a) の頂点で代わりに bootstrapping を行っても，制約条件を満たす状況を維持できる．これは，(a) の頂点で bootstrapping を行っても，(a) の出力する暗号文レベルは高くなるか変わらないため，(a) の他の出力先で bootstrapping を新たに実行する必要は生じないためである．

対して，(c) の頂点では同様の議論はできない．(c) の入力の 2 変数は直前のレベルが変化した親が異なるため，(c) で bootstrapping を行う必要があるときに，bootstrapping の位置をこれ以上親に動かすことができない．

このように，ある頂点から親を辿り続け，最初にレベルが変化しうる頂点を探すことを考える．そのようなレベルの変化しうる頂点が 1 つに定まるとき，親を辿る際に訪れた頂点の何れかで bootstrapping を行う代わりに最後にたどり着いた頂点で bootstrapping を行っても，復号の整合性を崩さず，ILP の制約を満たした状況を維持できる．よって提案手法では，以上の例に示した親子を辿る順序を逆にし，レベルが変化しうる頂点から探索を開始し，bootstrapping を行う位置を変更できる頂点の範囲を探索する．Bootstrapping を行う位置を動かせる範囲内の頂点同士をまとめることで，最終的に頂点数が削減される．

以上の頂点数削減は，幅優先探索を用いると実現可能であり，Algorithm 1 にアルゴリズムを示す．探索では順次，辺と頂点に番号を振る．ある頂点でレベルが変化しないとき，その頂点の入力と出力の辺には同じ番号を振り，レベルが変化する可能性があるならば，まだ使用していない番号を振る．辺と頂点に振られた番号は Algorithm 1 中の配列 num_E, num_V に相当し，変数 $nextID$ は相異なる番号を代入するため，未使用の最も小さい非負整数を保持している．この幅優先探索を用いたアルゴリズムは，時間計算量 $\mathcal{O}(|V|+|E|)$ で終了する．

すべての頂点に番号を振ったのち，同一番号の振られた頂点同士をまとめ，新たなグラフとして ILP に変換して厳密を求める．変換後のある番号の振られた頂点で bootstrapping を行うことは，最初のデータフローグラフにおいて該当の番号が最初に振られた頂点で bootstrapping を行うことに置き換えることができる．

Algorithm 1 幅優先探索による頂点数削減

Input: $G = (V, E)$ ▷ データフローグラフ
Output: $G' = (V', E')$ ▷ 頂点数削減後のグラフ

$nextID \leftarrow 0$
FIFO キュー $Q \leftarrow \phi$
$V' \leftarrow \phi$, $E' \leftarrow \phi$
配列 $num_V[]$, $num_E[]$ を 0 で初期化
for all $\{i \mid v_i \in V\}$ **do** ▷ 配列 $InCount$ の初期化
    $InCount[i] \leftarrow v_i$ に入力する辺の本数
end for
for all $\{i \mid v_i \in V$, v_i は乗算もしくは入力数 0 の頂点 $\}$ **do**
    Q に v_i を enqueue
    $V' \leftarrow V' \cup \{v_i\}$
    $num_V[i] \leftarrow nextID$
    $nextID \leftarrow nextID + 1$
    $InCount[i] \leftarrow 0$ ▷ v_i から探索させないため
end for
while $Q \neq \phi$ **do**
    $v_i \leftarrow Q$ から dequeue
    for all $\{(h, j) \mid e_h \in E, v_j \in V, e_h = (v_i, v_j)\}$ **do**
        ▷ v_i から出るすべての辺 $e_h = (v_i, v_j)$
        $num_E[h] \leftarrow num_V[i]$
        $InCount[j] \leftarrow InCount[j] - 1$
        if $InCount[j] = 0$ **then**
            Q に v_j を enqueue
            $S_j \leftarrow \{num_E[x] \mid e_x = (v_k, v_j) \in E, v_k \in V\}$
                ▷ S_j は v_j へ入力する辺に振られた番号の集合
            if S_j の全要素が同一の値 **then**
                $num_V[j] \leftarrow num_E[h]$
            else
                $V' \leftarrow V' \cup \{v_j\}$
                $num_V[j] \leftarrow nextID$
                $nextID \leftarrow nextID + 1$
            end if
        end if
    end for
end while
for all $\{(i, j) \mid e \in E, v_i, v_j \in V, e = (v_i, v_j)\}$ **do**
    ▷ 頂点削減後の辺を構成
    if $num_V[i] \neq num_V[j]$ **then**
        $E' \leftarrow E' \cup \{(v_{num_V[i]}, v_{num_V[j]})\}$
    end if
end for
return $G' = (V', E')$

4　評 価 実 験

4.1　実験回路と頂点数の削減

最初に，提案手法のアルゴリズムにより，計算回路の頂点数がどれだけ削減されるかを評価する．実験対象となる回路は, [12] に公開されているバイナリ回路のデータを用いた．

表 3 に用いた回路の特性と，提案手法の適用による頂点数の変化を示す．アルゴリズム上，準同型加算の頂点数のみ削減さ

表 3　回路の特性と頂点削減結果

回路の種類	$\|V\|$	$\|E\|$	乗算頂点数	加算頂点数 提案手法適用前	加算頂点数 提案手法適用後	頂点数削減率
adder_64bit	759	887	265	494	241	0.333
AES-non-expanded	33,616	33,872	6,800	26,816	25,132	0.050
comparator_32bit_signed_lt	300	364	150	150	60	0.300
DES-non-expanded	30,313	30,441	18,124	12,189	1,340	0.358
md5	77,861	78,373	29,084	48,777	14,631	0.439
mult_32x32	12,374	12,438	5,926	6,448	1,133	0.430
sha-1	106,601	107,113	37,300	69,301	24,627	0.419
sha-256	236,112	236,624	90,825	145,287	42,510	0.435

表 4　既存手法・提案手法における最適化に要する時間の計測結果（ミリ秒）

回路の種類	`Relin` ($k_r=1, k_m=1$)			`Relin` ($k_r=3, k_m=1$)			`Boot` ($L=2, N=2$)			`Boot` ($L=10, N=2$)		
	既存手法	提案手法	時間比	既存手法	提案手法	時間比	既存手法	提案手法	時間比	既存手法	提案手法	時間比
adder_64bit	4.18	2.87	0.69	4.20	2.88	0.69	8.90	6.35	0.71	31.54	15.62	0.50
AES-non-expanded	908.06	875.99	0.96	769.65	771.82	1.00	486.96	435.58	0.89	7526.93	7056.22	0.94
comparator_32bit_signed_lt	1.63	1.07	0.66	1.64	1.06	0.65	3.75	2.91	0.78	25.46	11.69	0.46
DES-non-expanded	207.16	124.27	0.60	207.42	124.48	0.60	411.39	268.00	0.65	2020.57	1083.89	0.54
md5	1560.78	1284.94	0.82	952.02	646.74	0.68	1234.34	737.02	0.60	5146.75	2521.92	0.49
mult_32x32	89.55	57.52	0.64	88.01	56.56	0.64	154.94	96.09	0.62	1994.23	1211.87	0.61
sha-1	2103.48	2157.49	1.03	1383.29	1073.85	0.78	1701.00	1044.52	0.61	10233.91	6410.72	0.63
sha-256	3712.55	2397.17	0.65	3701.76	2440.06	0.66	3819.26	2311.54	0.61	15342.90	8189.38	0.53

れる．本実験では，ほとんどの場合において，準同型加算の個数を約半数以下に削減し，頂点数全体でも約 3〜4 割の頂点を削減することができた．

4.2　頂点数削減による整数計画問題の高速化

次に，頂点数削減が，bootstrap problem や relinearize problem の ILP の計算時間にどの程度の影響を与えるか検証する．

ILP のソルバとして，Grobi Optimizer 8.0.1 [4]を用いた．実験・時間計測は，Intel Core i7-4790 CPU @3.60GHz，16GM of RAM，Ubuntu 18.04.1 LTS，gcc 7.3.0 のデスクトップコンピュータ上で行った．

それぞれの計算回路について，頂点数削減前のデータフローグラフを ILP に変換して解いたときの計算時間を既存手法，本提案手法による頂点数削減後のデータフローグラフを ILP に変換して解いたときの計算時間を提案手法とし，結果を表 3 にまとめる．なお，表における `Relin` は relinearize problem を，`Boot` は bootstrap problem を解くこと指しており，それぞれの場合において用いたパラメータを併記した．

実験結果より，提案手法を用いることでナイーブな既存手法と比較して多くの場合で最適化に要する時間を約 3〜4 割削減したことを確認できる．

5　おわりに

本稿では，bootstrap problem と relinearize problem の厳密解を高速に求める手法を提案した．厳密解を求める際に必要な ILP への問題変換の前にデータフローグラフに前処理を施すことで，考慮する必要のあるグラフの大きさを削減し，ILP の計算時間を削減した．評価実験では，多くの場合で，グラフの頂点数と ILP の計算時間をともに約 3〜4 割削減した．

今後の課題として，よりグラフの大きさを削減するアルゴリズムの検討，また，bootstrap problem や relinearize problem に対する精度の良い近似アルゴリズムの検討が挙げられる．

4：`http://www.gurobi.com/`

謝　辞

本研究は JST CREST Grant Number JPMJCR1503 の支援を受けたものである．

文　献

[1] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing, STOC '09*, pp. 169–178. ACM, 2009.

[2] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, Vol. 10, No. 4, pp. 283–424, 2016.

[3] Tancrède Lepoint and Pascal Paillier. On the minimal number of bootstrappings in homomorphic circuits. In *Financial Cryptography and Data Security: FC 2013 Workshops*, Vol. 7862, LNCS, pp. 189–200, 2013.

[4] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology – EUROCRYPT 2010, Proceedings*, Vol. 6110, LNCS, pp. 1–23, 2010.

[5] Hao Chen. Optimizing relinearization in circuits for homomorphic encryption. arXiv e-prints, October 2017.

[6] Fabrice Benhamouda, Tancrède Lepoint, Claire Mathieu, and Hang Zhou. Optimization of bootstrapping in circuits. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '17)*, pp. 2423–2433, 2017.

[7] Marie Paindavoine and Bastien Vialla. Minimizing the number of bootstrappings in fully homomorphic encryption. In *International Conference on Selected Areas in Cryptography*, Vol. 9566, LNCS, pp. 25–43, 2015.

[8] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations*

of secure computation, Vol. 4, No. 11, pp. 169–180, 1978.

[9] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pp. 97–106, 2011.

[10] Victor Shoup and Shai Halevi. HElib. `https://github.com/shaih/HElib`.

[11] 佐藤宏樹, 馬屋原昂, 石巻優, 山名早人. 完全準同型暗号による秘密計算回路のループ最適化と最近傍法への適用. *DBSJ Japanese Journal*, Vol. 16-J, No. 12, March 2018.

[12] Nigel P. Smart and Stefan Tillich. Circuits of basic functions suitable for mpc and fhe. `https://homes.esat.kuleuven.be/~nsmart/MPC/`.