DBSJ Japanese Journal
Vol. 16-J, Article No. 12, March 2018

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

Loop Circuit Optimization with Bootstrapping over Fully Homomorphic Encryption and its Application to Nearest Neighbor

佐藤 宏樹♡ 馬屋原 昂♡ 石巻 優♡
山名 早人♢

Hiroki Sato Akira UMAYABARA Yu ISHIMAKI Hayato YAMANA

近年，計算資源としてクラウドを使用するケースが増加する一方，第三者のサーバ上でデータ処理を行う場合は個人情報や機密情報の漏洩が懸念される．データの秘密を保持したまま解析を行う技術は秘密計算と呼ばれ，その **1** つに完全準同型暗号を用いた手法がある．完全準同型暗号では，計算を繰り返し行うために **bootstrap** と呼ばれる処理が必要となるが，**bootstrap** は膨大な計算量を要する．このため，ある計算のデータフローグラフが与えられたとき，**bootstrap** を行う回数を最小化することは全体の処理時間を削減することに直結する．**Bootstrap** を行う回数を最小化する問題に対する既存研究は対象のデータフローグラフを **DAG** に限定しており，ループにおけるイテレーション間の関係を扱うことができない．そこで，本稿ではループを含む計算回路において，ループアンローリングを用いながら全体の実行時間を最小化する **bootstrap** の配置決定手法を提案する．**2** 種類の回路に対して提案手法を適用し，評価実験を行った結果，イテレーションあたりの **bootstrap** 回数を，ナイーブな場合の **25%**〜**63%**に減らした．また，提案手法で得た結果を最近傍法へ適用し，ナイーブ手法と比較して **2.67** 倍の高速化を達成した．

These days, more people use cloud computing as a computationg resource, whereas it is concerned that processing data in a third-party environment can lead a leakage of private information. Processing data while keeping its secret is called secure computing, and one way to accomplish it is to use fully homomorphic encryption (FHE). To evaluate a complex circuit using FHE, running a time-consuming operation called bootstrap in some interval is required. Given a data flow graph, reducing the required number of bootstrap in the circuit directly leads to reduce the entire computation time of the circuit. In previous work to tackle the problems, objective circuits must be represented as a directed acyclic graph. Thus, the previous methods cannot handle loop-carried dependencies for the circuits containing any loop, which results in no optimization over iterations. In this paper, we propose a method to decide a near-optimal placement of bootstrap operations in a loop circuit by adopting loop unrolling technique. Compared to a naive method, our method successfully reduced the number of bootstrap operations per loop iteration up to 63 percent. We applied our method to a homomorphic nearest neighbor circuit, and achieved speeding up by 2.67 times.

♡ 非会員 早稲田大学大学院基幹理工学研究科
{hsato, uma, yuishii}@yama.info.waseda.ac.jp
♢ 正会員 早稲田大学理工学術院
yamana@waseda.ac.jp

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，保有するデータの利活用や解析の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技術は秘密計算と呼ばれる．本稿では特に完全準同型暗号（FHE: Fully Homomorphic Encryption）[1] を用いた秘密計算の研究を対象とする．FHE は暗号化状態で平文に対して加算と乗算に関する準同型性を持つ公開鍵暗号方式であるため，暗号化状態でもとの平文に対する演算を行える．しかし，FHE ではセキュリティ上の理由から暗号化の際に暗号文中にノイズが加えられる．ノイズは準同型演算を行うたびに増加し，一定量を超えると正しく復号できなくなる．この計算回数の限界という問題に対し，Gentry により提案された bootstrap 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，任意の回数の計算を暗号化状態で行うことができる．しかし，bootstrap は時間・空間計算量がともに膨大である．

これまで，準同型演算によるノイズの増加量を抑制する modulus reduction[2] や scale invariant 方式 [3] などの提案により，理論的な時間・空間計算量の削減が進んでいるものの，bootstrap の計算量は未だに実用的でない．したがって，既存の FHE を用いた応用研究では，bootstrap を用いないことで計算量を抑えた leveled HE 方式や somewhat HE 方式を用いた研究がほとんどである．しかし，これらの方式では正しく準同型演算を行える回数に上限があり，適用先が限られる．例えば機械学習アルゴリズムはループ処理を必要とするものが多く，一般に必要なループの繰り返し回数の見積もりが難しいため，bootstrap を備えた FHE を用いることになる．このとき，bootstrap の回数を減らすことは全体の計算時間を削減することに直結する．

計算回路のデータフローグラフ上のどの位置で bootstrap を行うかにより，必要となる bootstrap の回数が変化することが知られており，この回数を最小化する最適化問題は bootstrap problem と呼ばれる [4]．この問題は NP 困難であるが，問題を混合整数線形計画法に変換することで解を求める手法 [5] や，近似解を多項式時間で求める手法 [6] が提案されている．しかし，既存研究では対象とする計算回路をループ処理が含まれない DAG（有向非巡回グラフ）に限定しているため，ループのイテレーションを跨ぐデータ伝搬を考慮できない問題がある．また，暗号パラメータであるレベルを固定しているが，実際はレベルを変更すると準同型演算の計算時間が変わるため，最適なレベルの選択も考察する必要がある．

これらの問題に対して，本稿では FHE を用いた秘密計算におけるループ処理の最適化を行う．具体的には，ループ部のデータフローグラフにおいて，イテレーション 1 回あたりの bootstrap

回数が最小になるように bootstrap の位置を決定する．加えて，レベルを変化させると計算時間だけでなくセキュリティレベルも変化するため，予め設定した安全性を保つ範囲でレベルパラメータを変化させながら最適化を行う．一般にプログラム中での実行時間はループ処理がほとんどを占めるため，ループ部の最適化は効果が大きい．また，提案手法を FHE による最近傍法の秘密計算回路に適用し評価する．

以下，2 章で完全準同型暗号に関する背景を述べ，3 章で関連研究をまとめる．続く 4 章では本稿で提案する最適化手法について述べ，5 章で実験評価の方法と結果について述べる．最後に 6 章で本稿をまとめる．

2. 完全準同型暗号

完全準同型暗号（FHE, Fully Homomorphic Encryption）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 らが初めてこの概念を示し，当時のタイムシェアリングシステム上で，データの秘密を保持しながら計算を行うことについて問題提起した [7]．その実現は 30 年間，未解決であったが，2009 年の Gentry が FHE の具体的な構成法を発表 [1] したことを契機に FHE に関連した研究が活発となった．

2.1 Bootstrap

Gentry による FHE の構成のポイントは bootstrap と呼ばれる処理である．そもそも FHE は，暗号ベースにノイズを含むことにより実現される解読困難性を採用している．このノイズは暗号化時に最小で，加算や乗算を繰り返すたびに（特に乗算では初期の手法では指数的に，新しい手法でもほぼ線形に）増加する．そして，ノイズがある閾値を超えると暗号文を正しく復号できなくなる．この問題を解決するため，Gentry は bootstrap と呼ばれる，蓄積されたノイズを削減する手法を提案した．Bootstrap では，ノイズの蓄積した暗号文を再度暗号化し，「暗号化された秘密鍵」を用いて，準同型に復号処理を行う．この準同型に復号した結果として，復号に要したノイズのみが溜まった新たな暗号文を得ることができる．そして，復号回路に加えて 1 回以上のゲートの計算が可能であれば，任意の大きさの回路を準同型に計算できることになる．

2.2 BGV 方式

FHE の主要な方式の 1 つである BGV 方式 [8] について説明する．BGV 方式は FHE のオープンソースライブラリの `HElib` [1]が実装している暗号方式であり，本稿においても `HElib` を使用する．

BGV 方式は多項式環 $\mathbb{A} = \mathbb{Z}[x]/\Phi_m(x)$ 上で定義される．ここで，$\Phi_m(x)$ は m 番目の円分多項式である．平文空間は多項式環 $\mathbb{A}_t = \mathbb{A}/t\mathbb{A}$ で定義さる．この多項式環上で中国剰余定理を用いることで，1 つのベクトルを多項式上（1 つの平文空間）にパッキングすることが可能である（SV パッキング [9]）．

BGV 方式のパラメータは複数の整数の法の組 $q_L > q_{L-1} > \cdots > q_0$ が含まれ，これは modulus chain と呼ばれる．ここで，L はパラメータとして鍵生成時に渡すレベル（評価可能な回路の最大の深さ）を表す．暗号化されたばかりの暗号文の空間は $\mathbb{A}_{q_L} = \mathbb{A}/q_L\mathbb{A}$ を用いて定義され，準同型演算を繰り返すにつれ，順次小さな法へスイッチしていく．レベル i の暗号文の空間は $\mathbb{A}_{q_i} = \mathbb{A}/q_i A$ を用いて定義される．法を q_i から q_{i-1} へ置き換える操作（modulus switching）を行うと，暗号文に含まれるノイズを約 q_{i-1}/q_i の比で削減することができる．`HElib` は，乗算を行う毎に modulus switching を行うことで，正しく復号できる状態を保ちながら leveled HE を実現する．

[1] http://shaih.github.io/HElib/index.html

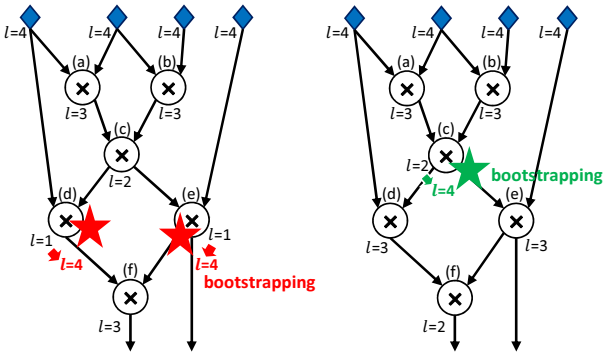

図 1: Bootstrap Problem の具体例（$L = 4, N = 4$）

Fig.1 An Example of Bootstrap Problem.

3. 関連研究

FHE において最も時間計算量の大きい処理は bootstrap である．例えば `HElib` を用いた場合，準同型な加算は数ミリ秒程度，乗算は数百ミリ秒程度であるのに対し，bootstrap には数分を要する．leveled FHE を用いた秘密計算において bootstrap を行う回数を最小化することは回路全体の高速化に直結する．Leveled FHE のレベルと計算回路のデータフローグラフが与えられたとき，回路中での bootstrap を行うべき回数を最小化する最適化問題は bootstrap problem と呼ばれ，研究が行われている．

3.1 概要

本節では bootstrap problem の既存研究の概要を述べる．データフローグラフ $G = (V, E)$ を，入力数が 0〜2 の頂点（ノード）の集合 V と，頂点間をつなぐ有向辺（エッジ）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．

Leveled FHE における暗号化直後の暗号文レベルを L，bootstrap を行った直後の暗号文レベルを N（正整数 $L \geq N > 0$）とする．Modulous switching を採用した leveled FHE では，計算回数に対するノイズの増加量は，加算は対数的，乗算は線形的である．特に，乗算を行うたびに事前に用意した暗号文空間の法を 1 段階小さなものへ切り替えるが，加算を行う際は法の切り替えは行わない [8]．用意する法は鍵生成時に決定するパラメータの組 $q_L > q_{L-1} > \cdots > q_1$ であり，レベル i の暗号文の法は q_i である．また，暗号文同士の準同型演算は同じレベル同士でのみ行えるため，異なるレベル同士の計算を行う場合は，最初にレベルの高い暗号文を低いレベルに合わせる操作を行う．

したがって，bootstrap problem では，準同型加算の出力暗号文のレベルは，2 つの入力暗号文のレベルの低いレベルに一致し，準同型乗算の出力暗号文のレベルは 2 つの入力暗号文のレベルの低いレベルからさらに 1 段階下がったものとする．そして，レベルが 0 以下になると正しく暗号文を復号できなくなるとする．

例えば，図 1 のデータフローグラフを考える．FHE のレベル設定は $L = N = 4$ とし，回路の入力が図の上部の菱形の頂点であり，上から下に向けて計算を行う．頂点横に記載された $L = x$ 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 bootstrap を行う」場合を考える（図左）．このとき，bootstrap 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 bootstrap を行う場合（図右），回路全体で bootstrap 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 らが 2013 年に初めて指摘した [4]．その後 2015 年に Paindavoine らがグラフ理論を用いた研究成果を発表した [5]．両者の研究では，bootstrap problem が NP 困難であると証明された．また，2017 年には Benehamouda

DBSJ Japanese Journal
Vol. 16-J, Article No. 12, March 2018

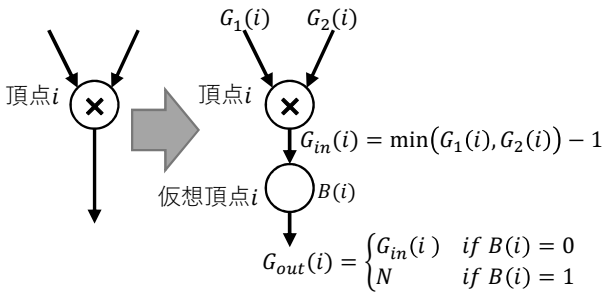

図 2: ゲートの定式化（乗算の場合）

Fig.2 Formulation of a Multiplication Vertex.

らは bootstrap problem の近似的な多項式時間の解法を示した [6]．本稿の提案手法は，Paindavoine らの手法を基にする．

3.2 Paindavoine らによる手法

本項では，Paindavoine らが提案した，問題を MILP の制約式へ変換する手法を説明する [5]．Paindavoine らは，bootstrap problem を混合整数線形計画法（MILP: Mixed Integer Linear Programming）に変換することで，ソルバを用いれば時間がかかるものの最適解を求めることが可能であることを示した．なお，本稿での記法に合わせるため，もとの式に変更を施した．

計算回路を表現するデータフローグラフ上の全ての頂点に対し番号を振る．i 番目の頂点（頂点 i）について，頂点が 2 入力とき，入力暗号文のレベルを $G_1(i)$ と $G_2(i)$ とし，1 入力のときはその入力暗号文のレベルを $G_1(i)$ とする．頂点 i の直後に，bootstrap を行う場所の候補となる頂点（仮想頂点 i と呼ぶ）を置く．仮想頂点 i に 0 または 1 の値 $B(i)$ を割り当て，$B(i) = 1$ のときのみ bootstrap を行う．頂点 i の出力暗号文のレベルは仮想頂点 i の入力暗号文のレベルと等しく，これを $G_{in}(i)$ とおく．仮想頂点 i の出力暗号文のレベルを $G_{out}(i)$ とする．$B(i) = 1$ の場合は $G_{out}(i) = N$ であり，$B = 0$ の場合は $G_{in}(i) = G_{out}(i)$ である．これらの変数の関連を図 2 にまとめる．

最適化の目的関数は，$B(i)$ が 1 となる個数を最小化すること，すなわち $\sum_i B(i)$ の最小化である．全ての頂点におけるレベル変化を，次の規則に従って線形制約式へ変換する．

一般制約：全ての暗号文レベルは 1 以上 L 以下であるため，式 (1) を得る．

$$\begin{cases} 1 \le G_1(i) \le L \\ 1 \le G_2(i) \le L \\ 1 \le G_{in}(i) \le L \\ 1 \le G_{out}(i) \le L \end{cases} \tag{1}$$

Bootstrapping：Bootstrapping を行う場合は $G_{out}(i) = N$ であり，行わない場合は $G_{out}(i) = G_{in}(i)$ であるため，式 (2) を得る．

$$G_{out}(i) = G_{in}(i)(1 - B(i)) + NB(i) \tag{2}$$

式 (2) を線形制約式にするため，十分大きな正整数 $X(\ge L)$ を用いて，式 (3) を得る．

$$\begin{cases} G_{out}(i) \ge NB(i) \\ G_{out}(i) \le N + X(1 - B(i)) \\ G_{out}(i) \ge G_{in}(i) - XB(i) \\ G_{out}(i) \le G_{in}(i) + XB(i) \end{cases} \tag{3}$$

加法：加法：出力レベルは，2 つの入力のうち低いレベルに一致するため，式 (4) を得る．

$$\begin{cases} G_{in}(i) \le G_1(i) \\ G_{in}(i) \le G_2(i) \end{cases} \tag{4}$$

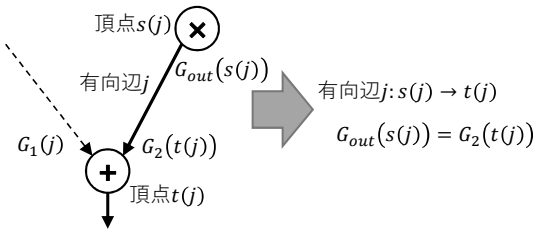

図 3: 有向辺の定式化

Fig.3 A Formulation of a Directed-edge.

乗法：出力レベルは，2 つの入力のうち低いレベルより 1 減るため，式 (5) を得る．

$$\begin{cases} G_{in}(i) \le G_1(i) - 1 \\ G_{in}(i) \le G_2(i) - 1 \end{cases} \tag{5}$$

1 入力頂点：出力レベルは，入力のレベルに一致するため，式 (6) を得る．

$$G_{in}(i) \le G_1(i) \tag{6}$$

有向辺：ある有向辺 j が頂点 $s(j)$ から頂点 $t(j)$ に存在するとする．このとき 2 頂点間の制約として，式 (7) を得る．この変数間の関係を図 3 にまとめる．

$$G_{out}(s(j)) = G_p(t(j)) \quad (p = 1 \text{ または } 2 \text{ の適する方}) \tag{7}$$

これら式 (1)〜式 (7) の制約式を全て満たす MILP の解に従って bootstrap を行えば，回路の出力結果は正しく復号できる．加えて，MILP の解が最適解であれば，bootstrap の回数も最小化されていることが保証される．

3.3 まとめ

FHE を用いた秘密計算回路における bootstrap 配置に関する既存研究では，対象とするデータフローグラフを DAG に限定している．また，回路がループを含む場合は，イテレーションの末尾で bootstrap を行うとした研究が存在する [10]．しかし，イテレーション間のレベルの変化を考慮した研究は行われていない．ループ伝搬依存を持つ変数のイテレーション間のレベル変化を考慮することで，計算回路全体で考えたときに必要な bootstrap の回数を減らすことが可能と考えられる．また，既存研究では，FHE のレベルを固定して最適化が行われているが，レベルが変化すると FHE の計算時間は変化するため，最適なレベル選択も課題である．

4. 提案手法

本研究では，3 章で述べた bootstrap problem の既存研究の問題である，ループを含む回路においてイテレーション間のループ伝搬依存を考慮した最適化を行えない問題の解決と，FHE のレベルパラメータを最適に選択することを目標とした提案を行う．なお，厳密な最適解を求めるには，後述のナイーブ手法 1 を適用する必要がある．したがって提案手法では，ナイーブ手法 1 よりも最適化で必要な計算量を抑えながらも，ナイーブ手法 1 の最適解に近い解を出力することを目標とする．

3 章で述べた既存手法のアルゴリズムは入力として DAG を受け取る．例えば，図 4 のループ回路を考える．この回路（データフローグラフ）において，変数 F_i と G_i はイテレーションごとに，ノイズの溜まっていない高いレベルの暗号文が新たに入力として与えられる．対して変数 x と y は，直前の繰り返しの出力が入力として与えられる．すなわち，x と y はループ伝搬依存を持つ変数であり，ループを跨ぐレベルの変化を考慮する必要がある．今

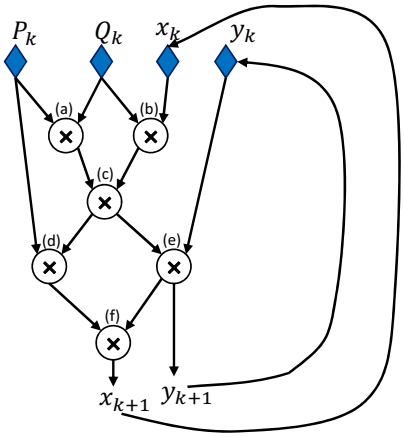

図 4: ループ回路の例

Fig.4 An Example of a Loop Circuit.

後，データフローフラフにおいて大文字で記す変数はループ伝搬依存を持たず，小文字で記す変数はループ伝搬依存を持つ変数とする．

このようなループを含む回路に対し，既存研究を直接用いて bootstrap の回数を減らす手法は次の 2 通りが考えられる．

ナイーブ手法 1： ループが無くなるまでデータフローグラフを繰り返し展開する．

ナイーブ手法 2： イテレーション 1 回のデータフローグラフで最適な bootstrap 配置を求め，求まった位置に加えてループ末尾で毎回 bootstrap を行う．

ナイーブ手法 1 では最適解を求めることが可能だが，繰り返し回数が増えるにつれてグラフが大きくなり，計算量も大きくなるため，繰り返し回数が大きい場合は MILP の計算時間が現実的ではなくなる．また，事前に繰り返し回数が決定されている必要があり，繰り返し回数が変わるたびに計算し直すことになる．

ナイーブ手法 2 では，1 の手法と比較して計算量が小さいが，求まる解が最適解であるとは保証されない．例えば，図 4 の回路に 2 の手法を適用すると，ループのたびに少なくとも (e) と (f) の計算後に bootstrap が必要である．しかし，これは変数 b に対して毎回 bootstrap をしているため，明らかに無駄がある．

そこで本稿では，ループを含む回路に適した bootstrap problem の解法を提案する．具体的に，次の 2 つの方針のもと手法を提案する．

1. ループアンローリングを用いて回路を展開し，イテレーションに跨った最適化を行う．
2. レベルパラメータを変化させ，最適なレベルパラメータを選択する．

提案手法により求まる解は最適でないが，良い精度で最適解（ナイーブ手法 1 の解）に近づくことを 5 節において実験結果から示す．

4.1 提案手法の適用範囲

提案手法ではループアンローリングにより，ループのイテレーション間の依存を解決し，bootstrap の配置を最適化する．ただし，対象とする計算回路は以下の条件を満たすとする．

- ループ依存距離が 2 以上のループ伝搬依存が存在しない．
- イテレーション内に条件分岐が存在せず，暗号文に対する計算は動的に変化しない．

また，提案手法は bootstrap の配置と FHE のレベル選択のみを最適化するため，以下の最適化は対象外とし，今後の課題とする．

- 回路を変形し，出力は同じだが，必要な bootstrap の回数が少ない回路へ変換する．
- 並列処理により全体での処理時間を小さくする．
- FHE のレベル以外のパラメータを最適に選択する．

4.2 既存研究のループへの対応（提案手法 1）

提案手法 1 では，Paindavoine らが提案した問題を混合整数計画法（MILP）に変換する手法 [5] を，ループのイテレーション間のデータ伝搬に対応させる．Paindavoine らの手法では，グラフの各頂点を通過する際の暗号文のレベルの変化と bootstrap の関係を定式化し，複数の制約式に変換し，MILP を解く．これを基に提案手法 1 では，ループを跨ぐ変数のレベルの変化がイテレーション間で伝搬するような制約式を加える．

提案手法 1 では加えて，MILP への定式化をループアンローリングに対応させる．ここで，ループの展開数に対してループ 1 回あたりの bootstrap 回数は単調減少しないが，$2n$ 回展開したときのループ 1 回あたりの bootstrap 回数は，n 回展開したときの回数以下であることは，周期性から明らかである．したがって，ループアンローリングを繰り返すことでループ 1 回あたりの bootstrap 回数を減らせると期待できる．

以下，提案する定式化手法を示す．

グラフ回路上の全ての頂点に対し番号を振る．データフローグラフのループ部を K 回分のイテレーションになるようにループを展開することを考える（K は 1 以上の自然数）．展開後のループ部において，もとのイテレーションの k 個目 ($1 \le k \le K$) における i 番目の頂点（頂点 (k,i) ）について，入力が 2 値のとき，入力暗号文のレベルを $G_1^k(i)$ と $G_2^k(i)$ とし，入力が単一の値のときはその入力暗号文のレベルを $G_1^k(i)$ とする．頂点 (k,i) の直後に bootstrap を行う場所の候補となる頂点が存在すると考える．この頂点に 0 または 1 の値 $B^k(i)$ を割り当て， $B^k(i) = 1$ のときのみ bootstrap を行う．頂点 (k,i) の出力暗号文のレベルは bootstrap を行う頂点の入力暗号文のレベルと等しく，これを $G_{in}^k(i)$ とおく．Bootstrap を行う頂点の出力暗号文のレベルを $G_{out}^k(i)$ とする．$B^k(i) = 1$ の場合は $G_{out}^k(i) = N$ であり， $B^k(i) = 0$ の場合は $G_{in}^k(i) = G_{out}^k(i)$ である．変数の割り当て例は図 2，図 3 を参照．すると，1 つのループのシーケンス内のレベル変化は，3.2 節における制約式（式 (1) ～式 (7)）を適用できる．これに加えて，ループを跨ぐレベルの伝搬を定式化する．

ループ伝搬依存として，頂点 $s(x)$ の出力が次のイテレーションの頂点 $t(x)$ に入力するとする．このとき，入出力のレベルが等しいことから，全てのループ伝搬依存について，式 (8) を得る．この変数間の関係を図 5 に示す．

$$G_{out}^k(s(x)) = G_p^{(k \bmod K)+1}(t(x)) \quad (p = 1 \text{ または } 2 \text{ の適する方}) \tag{8}$$

最適化問題の目的関数は， $B^k(i) = 1$ となる個数を最小化する

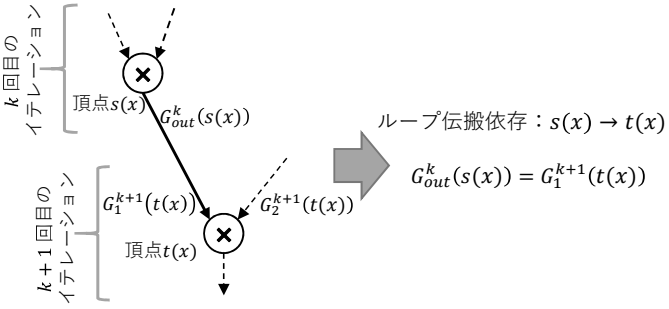

図 5: ループ伝搬依存の定式化

Fig.5 An Formulation of a Loop Carried Dependency.

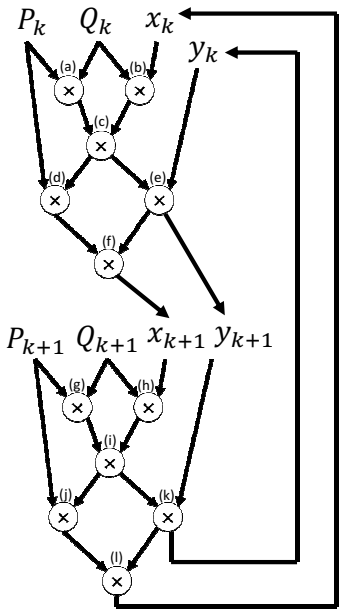

図 6: 回路を 2 回展開した例

Fig.6 An Example of Unfolded Two Iterations in a Loop.

こと，すなわち $\sum_k \sum_i B^k(i)$ の最小化である．

以上の定式化によって，ループを跨ぐレベルの変化とループアンローリングに対応する．例えば，図 4 の回路に対して本手法を適用し，もとのイテレーション 2 回分に展開した回路（図 6）において最適化を行うと，(c)，(f)，(j)，(k) の 4 か所で bootstrap を行う解が求まる．これは，もとのイテレーション 1 回あたり bootstrap を 2 回行うことを意味する．

提案手法で得た位置で bootstrap を行う回路では，与えた制約式より，回路のどの位置でも正しく復号ができると保証される．したがって,「展開したイテレーションの個数」で「繰り返し回数」を割り切れない場合も，提案手法は適用可能である．

ループ展開を繰り返すにつれて，MILP の制約式数と変数数が展開回数に比例して増加し，MILP の最適解を求める計算時間が指数的に増加する．そのため，本手法では，ループを展開して得られた結果と MILP に要した計算時間を観察し，繰り返し回数はヒューリスティックに決定する．

4.3 レベルパラメータの変化（提案手法 2）

FHE による秘密計算では，要求されるセキュリティレベルを担保した上で計算を行わなければならない．しかし，bootstrap 回数を減らすために暗号文レベルを高くすると保証されるセキュリティレベルが線形に減少するとともに準同型演算と bootstrap の計算時間は線形に増加する．すなわち，bootstrap 回数を減らすだけでは，セキュリティを担保しつつ回路全体の計算時間を最小化することができない．場合によっては，bootstrap 回数が増えたとしても暗号文レベルを低くした方が高速化できる．

そこで提案手法では，暗号文レベルを除いたパラメータセットと最低限要求されるセキュリティが与えられたとき，セキュリティ要件を満たす範囲内で暗号文レベルを変化させながら，前項の手法を繰り返し適用する．

暗号文の初期レベル L ごとに，乗算と bootstrap の 1 回あたりの実行時間 t_{mul}, t_{bs} を予め計測することにより，各レベル採用時のループ 1 回あたりの回路の計算時間 t_{total} を式 (9) で求めることができる．ここで，データフローグラフ中の乗算ゲート数を n_{mul}，そのレベルにおける bootstprap 回数の，もとのイテレーション 1 回あたりの最小値を n_bs とする．したがって，提案手法 2 では，式 (9) の t_{total} を最小とする暗号文の初期レベル L を選択する．

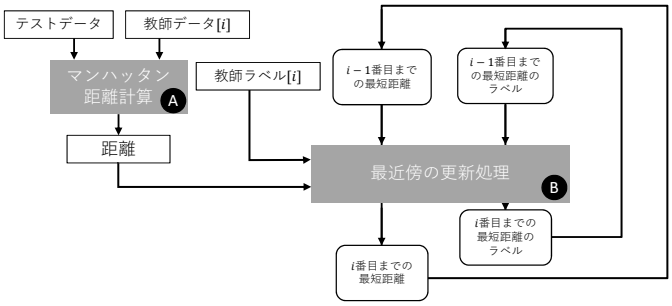

図 7: 最近傍法の処理の流れ

Fig.7 An Overview of Nearest Neighbor.

$$t_{total} = t_{bs} \times n_{bs} + t_{mul} \times n_{mul} \quad (9)$$

ここで，乗算と bootstrap の計算時間のみを考慮した．これは，例えば HElib を用いた場合，bootstrap が 2〜5 分，乗算が数百ミリ秒かかるのに対し，加算は数ミリ秒程度であり bootstrap と比べて無視できるほど短いためである．

5. 評価実験

本節では，提案手法の評価実験とその結果と考察を述べる．

5.1 実験回路

本研究では，2 種類の回路を実験のために用意した．

1 つ目の回路は，4 章の提案手法での例示に用いた図 4 の回路（小回路と呼ぶ）である．

2 つ目の回路は，最近傍法による識別器の回路である．最近傍法による識別では，テストデータに対して全ての教師データとの距離を繰り返し計算するループが存在する．したがって，最近傍法の評価回路のループ部に対して提案手法を適用できる．最近傍法の計算の流れを図 7 に示す．処理は大きく 2 つに分けることができ，前半が「テストデータと教師データの距離計算」（図中 A），後半が「最近傍の更新計算」（図中 B）である．前半はイテレーション間にデータ依存が無いため並列に計算できる．対して後半はイテレーション間にデータ依存（図 7 における「最短距離」と「最短距離のラベル」に相当する変数）が存在する．本稿では，後半の最近傍の更新処理のみを回路（最近傍更新回路と呼ぶ）で表し，最適化を行う[2]．ここで，FHE で設定するレベルパラメータによって距離計算の出力暗号文のレベルが変化する．したがって，距離計算の最後に bootstrap を行い距離計算の出力暗号文のレベルを N に統一することで，最近傍更新回路の最適化における距離計算の影響を排除する．

本稿では実験 3 で，最近傍法の準同型演算を行う．実験データには MNIST の手書き数字画像データ[3]を用いる．本実験では元のデータに対して主成分分析により次元を削減し，その上位 32 次元成分（1 次元は 8bit）を用いる．距離の計算は，準同型演算が複雑にならないマンハッタン距離を用いる．暗号文にデータをバイナリ表現でパッキングし，SIMD 演算を利用した計算回路を組んだ．

本実験で対象とする回路の特性を表 1 にまとめる．なお，表中の「その他」のゲートとは，定数との加乗算やパッキングに対するシフト演算など，レベルに変化がなく，1 入力 1 出力のゲートをまとめたものである．また，最近傍更新回路で想定するループ

[2]処理の並列化による処理の高速化は本稿の対象外である．本来は前半の距離計算を含めた回路を最適化対象とすべきだが，対象回路が大きくなり計算に時間がかかるため，最近傍更新回路のみに対する最適化を考える．

[3]THE MNIST DATABASE of handwritten digits http://yann.lecun.com/exdb/mnist/

回数が少ない理由は，既存手法 1 を適用する際に MILP の計算に時間がかかるためである．

5.2 実験環境

本実験は 2 種類の計算機上で行う．それぞれを構成 1，構成 2 とし，以下に具体的な構成を示す．

構成 1 72Core (Intel Xeon E7-8880 v3 @ 2.30GHz × 4), 1TB メモリ搭載サーバ

構成 2 28Core (Intel Xeon E5-2690 v4 @ 2.60GHz × 2), 500GB メモリ搭載サーバ

5.3 実験 1（提案手法 1）

実験 1 では，前項で説明した小回路と最近傍更新回路に対し，4 章で示したナイーブ手法 1，ナイーブ手法 2 と提案手法 1 をそれぞれ適用し，提案手法 1 を評価する．ナイーブ手法 1 ではループを繰り返す回数だけ展開したグラフに対して 3.2 節の既存研究を適用するため，最適解が求まる．ナイーブ手法 2 では 1 回のイテレーションに 3.2 節の既存手法を適用して得た解に加え，ループの末尾で bootstrap を必ず行う．

レベルパラメータは最後の最近傍法の実験で用いる FHE パラメータで設定可能なレベルから一部を選択した．MILP のソルバは gurobi optimizer 7.0.1[4] を利用する．

実験結果として，各手法での展開前のイテレーション 1 回あたりの bootstrap の回数を比較する．小回路における結果を表 2 に，最近傍更新回路における結果を表 3 に示す．表中の太字下線で示す値の箇所は，そのレベルパラメータにおいて，実験した範囲内でイテレーションあたりの bootstrap の必要回数が最も少ない箇所（複数ある場合は展開数の少ない箇所）である．また，「提案手法 1 ÷ナイーブ手法 1」の行は，各レベルパラメータにおける提案手法 1 とナイーブ手法 1 との bootstrap 回数の比であり，提案手法 1 で求まる回数が，ナイーブ手法 1 で求まる回数にどれだけ近づいたかを示す．「提案手法 1 ÷ナイーブ手法 2」の行は，提案手法 1 とナイーブ手法 2 との bootstrap 回数の比であり，ナイーブ手法 2 と比較した際の提案手法 1 の bootstrap 回数の割合を示す．

どちらの回路の場合でも，提案手法 1 では，展開数を増やすことでループ 1 回あたりの bootstrap 回数を削減でき，最適解と考えられるナイーブ手法 1 の値に少ない展開回数で近づく．MILP の計算量は展開数に対し指数的に大きくなるため，展開数が少ない方が望ましい．提案手法 1 では，高々7〜8 回の展開数で最適解の約 1.1 倍〜1.2 倍の解を得ることができた．また，提案手法 1 で最小の bootstrap 回数は，ナイーブ手法 2 で必要な bootstrap 回数の 25%〜63%に減少した．

5.4 実験 2（提案手法 2）

実験 2 では，5.3 節の実験結果を用いて，各レベルパラメータにおける，回路の展開前のイテレーション 1 回あたりの実行時間を比較する．5.3 項の実験 1 で求めた各レベルパラメータにおけるイテレーション 1 回あたりの bootstrap 回数の最小数（表では「実験 1 の結果」と表記）を用いて，それぞれのレベルでのイテレーションの平均計算時間を式 (9) に基づき概算した．準同型乗算と bootstrap にかかる時間は，`HElib` で表 4 に示すパラメータを設定して構成 2 の計算機上で 5 回実行した平均を利用する．

小回路と最近傍更新回路のそれぞれに対する提案手法 2 の適用結果を表 5 と表 6 にまとめる．

表 5 では，$(L, N) = (20, 9)$ のときと $(18, 7)$ のときで最適解が等しくなり，回路全体の計算時間は $(18, 7)$ の方が小さい．つまり，$(20, 9)$ のときと比較して，セキュリティも計算時間も $(18, 7)$ のパラメータのときが優れる．しかし，表 6 ではレベルが高いほど計算時間が小さくなっている．一般にはレベルが高いと bootstrap

[4] `http://www.gurobi.com/`

表 1: 実験対象回路の特性

Table1 Specifications of Target Circuits

	ゲート数				ループ依存を持つ変数数	最大繰り返し回数
	乗法	加法	その他	合計		
小回路	6	0	0	6	2	50
最近傍更新回路	11	8	9	28	2	18

表 2: イテレーションあたりの bootstrap 回数（小回路）

Table2 The Number of Bootstrappings in a Toy Circuit.

パラメータ (L, N)		(18,7)	(20,9)	(22,11)	(24,13)	(26,15)	(28,17)
ナイーブ手法 1 (ループ 50 回)		0.92	0.82〜0.92 ※	0.68	0.60	0.52	0.44
ナイーブ手法 2		2.00	2.00	2.00	2.00	2.00	2.00
提案手法 1 (最適化対象回路の展開したイテレーション数)	1	2.00	2.00	2.00	2.00	2.00	2.00
	2	1.50	**1.00**	1.00	1.00	1.00	1.00
	3	**1.00**	1.00	1.00	1.00	0.67	0.67
	4	1.25	1.00	**0.75**	0.75	0.75	**0.50**
	5	1.20	1.00	0.80	0.80	0.60	0.60
	6	1.00	1.00	0.83	0.83	0.67	0.67
	7	1.14	1.00	0.86	**0.71**	**0.57**	0.57
	8	1.13	1.00	0.75	0.75	0.63	0.50
	9	1.00	1.00	0.78	0.78	0.67	0.56
	10	1.10	1.00	0.80	0.80	0.60	0.60
提案手法 1 ÷ ナイーブ手法 1		1.09	1.22〜1.09	1.10	1.18	1.10	1.14
提案手法 1 ÷ ナイーブ手法 2		0.50	0.50	0.38	0.36	0.29	0.25

表 3: イテレーションあたりの bootstrap 回数（最近傍更新回路）

Table3 The Number of Bootstrappings in a NN Circuit.

パラメータ (L, N)		(22,11)	(24,13)	(26,15)	(28,17)	(30,19)
ナイーブ手法 1 (ループ 18 回)		1.11	1.06〜0.89 ※	0.89	0.67	0.56
ナイーブ手法 2		2.00	2.00	2.00	2.00	2.00
提案手法 1 (最適化対象回路の展開したイテレーション数)	1	2.00	2.00	2.00	2.00	2.00
	2	1.50	1.5	**1.00**	1.00	1.00
	3	1.33	1.33	1.00	1.00	1.00
	4	**1.25**	1.25	1.00	**0.75**	0.75
	5	1.40	1.20	1.00	0.80	0.80
	6	1.33	1.17	1.00	0.83	**0.67**
	7	1.29	1.14	1.00	0.86	0.71
	8	1.25	**1.13**	1.00	0.75	0.75
提案手法 1 ÷ ナイーブ手法 1		1.13	1.27〜1.07	1.12	1.12	1.20
提案手法 1 ÷ ナイーブ手法 2		0.63	0.57	0.50	0.38	0.34

- ※：構成 1 の計算機上で MILP の計算を実行したが，解が収束せず約 2 日経過でメモリ不足となった．
- （注）ナイーブ手法 2 と提案手法のイテレーション数 1 はどちらもループを展開しない点は共通である．しかし，ナイーブ手法 2 はループ依存の伝搬を行わないのに対し，提案手法ではループ依存の伝搬を行う点が異なる．

表 4: HElib で用いたパラメータ

Table4 HElib Parameters we used.

パラメータ	値
平文空間の法 p	2
円分多項式の次数 m	31775
SV パッキングのスロット数	1200

表 5: 小回路の最適化（提案手法 2）

Table5 Result of Optimization (Toy Circuit)

パラメータ (L,N)	(18,7)	(20,9)	(22,11)	(24,13)	(26,15)	(28,17)
セキュリティ [bit]	180.7	146.6	120.7	107.4	90.9	80.7
乗算実行時間 [sec]	0.143	0.162	0.158	0.165	0.168	0.191
Bootstrap 実行時間 [sec]	77.27	82.89	85.00	86.29	86.50	90.38
実験 1 の結果（表 2）[# of bootstrap/iter.]	1.00	1.00	0.75	0.71	0.57	0.50
1 イテレーション概算実行時間 [sec/iter.]	78.13	83.86	64.70	62.26	50.31	46.34

表 6: 最近傍更新回路の最適化（提案手法 2）

Table6 Result of Optimization (NN Circuit)

パラメータ (L,N)	(22,11)	(24,13)	(26,15)	(28,17)	(30,19)
セキュリティ [bit]	120.7	107.4	90.9	80.7	65.4
乗算実行時間 [sec]	0.158	0.165	0.168	0.191	0.190
Bootstrap 実行時間 [sec]	85.00	86.29	86.50	90.38	93.25
実験 1 の結果（表 3）[# of bootstrap/iter.]	1.25	1.13	1.00	0.75	0.67
1 イテレーション概算実行時間 [sec/iter.]	107.99	99.32	88.35	69.89	64.57

の回数が減り全体の計算時間も短くなるが，回路や FHE のパラメータに依存して，レベルを高くしても全体の計算時間が短くならない場合があることが本実験により確かめられた．

5.5 実験 3（最近傍法への適用）

3 番目の評価実験では，FHE を用いた最近傍法による識別器の準同型評価を実装し，計算時間を計測する．提案手法により bootstrap 配置の最適化を施した最近傍法と，ナイーブ手法 2 による最近傍法を実装し，計算時間を比較する．セキュリティレベルは 80bit 以上を確保する．本実験では，1 つのテストデータ当たり 72 件の教師データと距離を計算して最近傍を求める．ここで，平文ベクトルに対して SV パッキングによりビット単位暗号化を行い，1 つの暗号文に 4 件のデータを詰め込む．したがって，実際には 18 回のイテレーションを実行する．

提案手法の評価のため，事前に全ての教師データとテストデータとの距離を計算しておき，ループ部分である最近傍更新回路に要する時間のみを計測する．

本実験は構成 2 の計算機上で行い，HElib は 5.4 節と同じ表 4 のパラメータを用いる．

提案手法による最適化は，表 6 の結果より，全体の計算時間が最も小さくなるレベルパラメータ $(L,N)=(28,17)$ を選択し，MILP により得た解の位置で bootstrap を行う．ナイーブ手法 2 でのレベルパラメータは，提案手法と同じ (28, 17) で実験を行う．また，比較のため，bootstrap 配置の最適化を行わず，計算後にレベルが 1 となるたびに bootstrap を行う方法を「ナイーブ手法 3」と定義し，ナイーブ手法 3 の場合も同様にして計算に要する時間を計測する．

この実験結果を表 7 にまとめる．参考として，テストデータと全ての教師データとの距離を並列に計算するのに要した時間を「距離計算時間」とした．速度向上率はナイーブ手法 2 を基準とし，

表 7: 最近傍更新回路の準同型評価時間

Table7 Computation Time to Evaluate Hom. NN

最適化手法	ナイーブ手法 2	ナイーブ手法 3	提案手法	速度向上率
（参考）距離計算時間 [sec]	1,292	1,716	1,310	-
最近傍更新回路実行時間 [sec]	3,365	2,989	1,262	2.67
イテレーションあたり最近傍更新回路の実行時間 [sec/iter.]	187.0	166.1	70.1	2.67

「ナイーブ手法 2 の所要時間÷提案手法の所要時間」を計算した．

表 7 より，提案手法はナイーブ手法と比較して 2.67 倍の高速化を達成できたことがわかる．具体的に bootstrap の回数を比較すると，ナイーブ手法 2 ではイテレーションあたり 2 回であり，提案手法ではイテレーションあたり 0.75 回であり，この比は $2 : 0.75 \simeq 2.67 : 1$ である．これは，ナイーブ手法 2 に対する提案手法の速度向上率に一致する．計算速度の向上率と bootstrap 回数の減少率が一致しているため，bootstrap の回数を減らしたことが実行時間の削減に直結したことがわかる．

また，ナイーブ手法 3 は，ナイーブ手法 2 よりも実行時間が短くなった．これは，ナイーブ手法 2 ではイテレーション毎に bootstrap を行う必要が無い箇所でもイテレーション毎に必ず bootstrap を行う無駄があったためである．ナイーブ手法 3 では，この無駄がなくなった．しかし，レベルが 1 にならないと bootstrap を行わないため，回路全体で見たときの最適な bootstrap の位置は考慮されていない．したがって，ナイーブ手法 3 は提案手法より実行時間が長い．

5.6 追加実験と考察

本節では，提案手法に対する考察を行う．考察では，回路の例として，5.1 章で述べた回路に加えて，図 8 の回路 (a)，(b) を例に用いる．この 2 つの回路に対し提案手法 1 を適用した結果を表 8 に示す．最大 20 回まで展開を行い，イテレーションあたりの bootstrap 必要回数の最小値と，その最小値を得る展開数の最小数をまとめる．なお，レベルパラメータは例示のために小さな値を選択した．

ループ伝搬依存を持つ変数の個数（N_{props} とおく）について考察する．小回路と最近傍更新回路はともに $N_{props}=2$ であった．対して，図 8 の回路 (a)，(b) は $N_{props}=4$ である．ここで，ナイーブ手法 2 では，ループ伝搬依存を持つ変数はイテレーション末尾で必ず bootstrap を行うため，図 8 の回路 (a)，(b) は，イテレーションごとに最低でも 4 回の bootstrap を必要とする．対して提案手法を適用した場合，回路 (a) の場合最大 1.6 回，回路 (b) の場合 1.0 回であり，回路全体の計算時間に換算すると 2.5

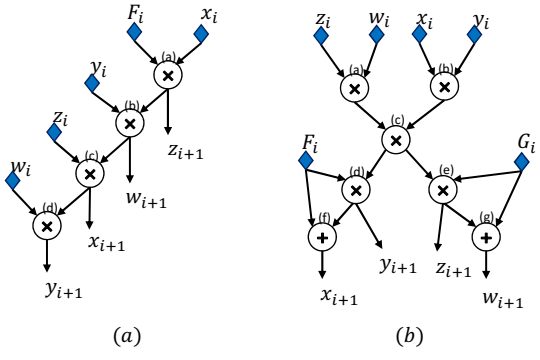

図 8: 回路例

Fig.8 Examples of a Circut.

表 8: 図 8 (a), (b) の回路への提案手法適用結果

Table8 Result of Optimization to Citcuits in Fig.8

対象回路	図 8 (a)				図 8 (b)				
パラメータ (L, N)	(4,4)	(5,5)	(6,6)	(7,7)	(3,3)	(4,4)	(5,5)	(6,6)	(7,7)
イテレーションあたりの bootstrap の必要回数の最小値	1.60	1.00	0.86	0.67	1.00	1.00	1.00	1.00	0.5
解を得た最小展開数	5	4	7	3	1	1	1	1	2

〜4 倍の高速化となる結果を得た．ナイーブ手法 2 ではイテレーションごとに N_{props} 回の bootstrap が要求されるため，N_{props} が大きいほど，ナイーブ手法 2 と比較したとき，提案手法による高速化の効果が大きい．

次にループアンローリングを行う展開数について考察する．例えば表 3 のレベルパラメータ (24, 13) では，展開数の増加に対してイテレーションあたりの bootstrap の回数の減少量は小さい．展開数が多くなると，解は下界に漸近してゆくと考えられるため，展開数を大きくしても最適化効果が小さいといえる．5.3 章〜5.5 章での実験結果でも，展開数が高々7 回で十分な解を得た．一般に，ループアンローリングにより展開するイテレーション数は 7,8 程度の少ない回数で十分であると考えられる．

以上の考察より，提案手法は，ループ伝搬依存を持つ変数の個数が多いほど，ナイーブ手法と比較して良い解を得る．また，ループアンローリングにより展開するイテレーションの個数は，小さくてよい．

6. おわりに

本稿では，完全準同型暗号を用いた秘密計算回路は bootstrap が計算時間のボトルネックとなることに注目し，bootstrap の必要回数を最小化する bootstrap problem の研究を行った．Bootstrap problem の既存研究では対象のデータフローグラフを DAG に限定し，ループ回路におけるイテレーション間の関係を扱うことができない．そこで本稿では，最適化対象とする回路がループを含む場合の最適化手法を提案した．提案手法では，既存研究をループ伝搬依存に対応するように変更を加え，ループアンローリングを繰り返す．2 種類の回路に対して提案手法を適用し，評価実験を行った結果，イテレーションあたりの bootstrap 回数を，ナイーブな場合の 25%〜63%に減らした．提案手法で得た結果を最近傍法へ適用し，ナイーブ手法と比較して 2.67 倍の高速化を達成した．また，様々なレベルで解を求め，レベルが低くても全体の実行時間は短くなる場合がありえることを実験結果より示した．

今後の研究課題として，1) 複数の bootstrap を並列に同時計算する場合は全体の実行時間を最小化する bootstrap 配置が変化する可能性があるため，並列計算に対応した bootstrap problem の研究，2) 本稿ではアンローリング回数やレベルを全探索したが，全探索せずに最適なパラメータと bootstrap 配置を求める手法の研究，3) 回路を変形し，出力は同じだが必要な bootstrap の回数が少ない回路へ変換する手法の研究，などが考えられる．

[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 *Proc. of Ann. ACM Symp. on Thory of Computing (STOC 2009)*, Vol. 9, pp. 169–178, 2009.

[2] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *Proc. of the 52nd Ann. Symp. on Foundations of Computer Science (FOCS 2011)*, pp. 97–106, 2011.

[3] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in Cryptology–CRYPTO 2012, LNCS*, Vol. 7417, pp. 868–886, 2012.

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

[5] Marie Paindavoine and Bastien Vialla. Minimizing the number of bootstrappings in fully homomorphic encryption. In *Proc. of the 22nd Int'l Conf. on Selected Areas in Cryptography, SAC 2015, LNCS*, Vol. 7839, pp. 25–43, 2015.

[6] Fabrice Benhamouda, Tancrède Lepoint, Claire Mathieu, and Hang Zhou. Optimization of bootstrapping in circuits. In *Proc. of the 28th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pp. 2423–2433, 2017.

[7] 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.

[8] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proc. of the 3rd Innovations in Theoretical Computer Science Conference*, pp. 309–325, 2012.

[9] Nigel P Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Designs, codes and cryptography*, pp. 1–25, 2014.

[10] Yu Ishimaki, Hiroki Imabayashi, Kana Shimizu, and Hayato Yamana. Privacy-preserving string search for genome sequences with fhe bootstrapping optimization. In *Proc. of the 2016 IEEE Int'l Conf. on Big Data, IEEE BigData 2016*, pp. 3989–3991, 2016.

佐藤 宏樹 Hiroki SATO
平成 29 年早稲田大学基幹理工学部卒．同大学基幹理工学研究科・情報理工情報通信専攻・修士課程在学中．

馬屋原 昂 Akira UMAYABARA
平成 28 年早稲田大学基幹理工学部総代卒．同大学基幹理工学研究科・情報理工情報通信専攻・修士課程在学中．

石巻 優 Yu ISHIMAKI
平成 27 年早稲田大学基幹理工学部卒．同大学基幹理工学研究科・情報理工情報通信専攻・博士後期課程在学中．暗号技術を用いた秘匿計算の高速化の研究に従事．

山名 早人 Hayato YAMANA
昭和 62 年早大・電子通信学科卒．平 5 年博士後期課程了．博士 (工学)．平 5〜12 年電子技術総合研究所．平 12 年早大助教授．平 17 年同教授，現在に至る．大規模データ解析研究に従事．