Vol.2016-CSEC-74 No.38
Vol.2016-SPT-19 No.38
2016/7/15

秘密計算フィッシャー正確検定 (2) ～標本数が多い場合

濱田 浩気[1,a)]　長谷川 聡[1]　千田 浩司[1]　荻島 創一[2,3]　三澤 計治[2,3]　長崎 正朗[2,3]

概要：ゲノム解析等でしばしば用いられるフィッシャー正確検定を秘密計算で効率的に実現する手法を提案する．フィッシャー正確検定は階乗計算 (または対数計算) を繰り返し行うため処理コストが大きく，通常の計算よりも一般に膨大な処理を伴う秘密計算で実現する場合の処理コスト削減は大きな課題と言える．筆者らの別の結果では，標本数が少ないことを前提としていた．本稿では，標本数が多い場合でも効率的なアルゴリズムを提案する．標本数を N とすると，素朴な方法では $\Omega(N^2)$ の通信量を必要とするが，本稿で提案するアルゴリズムは通信量が $O(N)$ である．

Privacy Preserving Fisher's Exact Test(2) - For Large Samples

Koki Hamada[1,a)]　Satoshi Hasegawa[1]　Koji Chida[1]　Soichi Ogishima[2,3]　Kazuharu Misawa[2,3]
Masao Nagasaki[2,3]

1. はじめに

近年の技術発展でゲノム解析のコストは大幅に下がり，盛んに行われるようになってきた．これにより遺伝子の個人差と疾患の関係が明らかになってきている．複数の研究機関が保有するゲノムデータを横断的に分析することで，より大規模に分析を進めたいというニーズも高まっている．そこで，複数の研究機関が，互いにゲノムデータを開示することなく，暗号化したままで比較し，ゲノム解析を可能にする手法の開発を目指す．

情報を公開することなく分析を可能にする手法として，秘密計算が知られている．秘密計算は Yao による基本的なアイディア [20] を端緒とする技術であり，暗号化などの方法でデータを秘匿化したまま一度も元のデータに戻すことなく任意の計算を行う．秘密計算を用いることで通常のデータ分析と同等の結果を高いプライバシ保護の下で得ることができるが，秘密計算では通常の計算機上の計算に比べて処理速度が低下してしまうことが実用上の課題となっている．その原因は大きく二つある．

処理速度の低下の大きな要因の一つは基本演算のオーバーヘッドである．マルチパーティ計算ではデータの秘匿性を保つために，乗算のような通常の計算機では一命令で実行可能な基本演算にも複雑な処理を必要とし，その結果，全体の処理時間も大きくなってしまう．これに対しては近年改良が進んでおり，Bogdanov らによる Sharemind[5]，Ben-David らによる FairplayMP[3]，Burkhart らによる SEPIA[6]，Geisler による VIFF[10]，Henecka らによる TASTY[15]，Chida らによる MEVAL [7] など，効率のよい基本演算を備えたフレームワークの提案，実装が行われている．

もう一つは，回路に基づいた構成による計算量の悪化である．秘密計算は Goldreich ら [12] や Ben-Or ら [4] により提案された回路に基づく構成方法を用いることで，任意の計算を実現することができる．この構成方法では，所望の計算を論理回路で表現し，秘密計算上の基本的な演算の組み合わせで論理回路を模倣することにより任意の計算をデータを秘匿したまま行う．しかしながら，この回路に基づいた一般的な構成方法を用いると，実用上重要な多くのアルゴリズムで通常の計算機上での計算に比べて計算量が

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories, 3-9-11, Midori-cho, Musashino-shi, Tokyo 180–8585, Japan
[2] 東北大学東北メディカル・メガバンク機構
Tohoku Medical Megabank Organization, Tohoku University, 2-1, Seiryo-machi, Aoba-ku, Sendai 980-8573, Japan
[3] 東北大学大学院医学系研究科
Graduate School of Medicine, Tohoku, University, 2-1, Seiryo-machi, Aoba-ku, Sendai 980-8575, Japan
[a)] hamada.koki@lab.ntt.co.jp

大きくなってしまう．基本演算の効率化は定数倍の改善に過ぎず，多量のデータを扱うためには，計算量の悪化の解決が不可欠である．このため，特定の処理ごとに秘密計算上の効率的なアルゴリズムを設計する研究が進んでおり，Aggarwal らによる k 番目の要素の選択 [1] や Damgård らによるビット分解 [8]，Nishide と Ohta による比較 [19]，Ning と Xu による剰余計算 [18]，Goodrich によるソート [14] などが提案されている．

秘密計算の効率化に伴って，実用的な演算も実現されるようになってきた．近年では χ^2 検定，Cochran-Armitage 検定，transmission disequilibrium 検定 [16] なども行われてきている．

ゲノム統計学の中でも，特定の形質と関連のあるゲノム変異を特定する全ゲノム関連解析 (genome-wide association study; GWAS) は最も一般的なアプローチの一つである．フィッシャー正確検定 [9] は，一塩基多型 (single-nucleotide polymorphism; SNP) と呼ばれるゲノム塩基配列中の変異が見られる箇所の塩基と特定の病気のような形質との関係を確かめるのに使われ，GWAS において基本的かつ重要な関数である．Lu ら [17] はフィッシャー正確検定の近似法である χ^2 検定の計算を安全に外部委託する実験を行っている．一方，Yates により通常の χ^2 テストは期待値が十分に小さいときにエラーが大きくなりがちであることが示されている [21]．

フィッシャー正確検定では正確な確率が計算される反面，多くの計算資源を必要とする．そのため，実行時間を短縮するために表参照のような実装の技法が用いられる．秘密計算で表参照を適用しようとする場合，素朴な方法を使うと表参照の度に表全体へアクセスする必要があり，非常に効率が悪い．表参照を効率的に行う手法としては Oblivious RAM (ORAM) [13] と呼ばれる，1 回あたりのアクセスに要する通信量がメモリサイズの線形より小さくなる方法が研究されている．しかしながら，同時のメモリアクセスを行うことができる手法は知られておらず，複数要素の読み込みを行うためには続けて複数回のアクセスを行う必要があり効率が悪い．

1.1 貢献

本稿では，GWAS で必要とされているフィッシャー正確検定を秘密計算で効率よく実現する手法を提案する．標本数の上界を N とするとき，単純な方法では $\Omega(N^2)$ の通信量を要するが，提案手法は $O(N)$ で実現する．提案手法は通信量が標本数 N に対して線形で漸近的に効率がよいため，N が大きくなる場合にも効率的である．

提案アルゴリズムの効率的な通信量は n 要素の秘匿文の配列から位置を明かさずに m 個の要素を取り出す通信量が $O(n)$ の一括読み込みアルゴリズムによる．先行研究として濱田ら [22] による $O((m+n)\log(m+n))$ のアルゴリズムが提案されているが，本稿では m 個の要素の相対位置が公開できる場合に限定し，$O(n)$ の通信量で m 要素の読み込みを行うアルゴリズムを提案する．

表 1 分割表の例

	塩基		
病気	A	a	病気による小計
あり	a	b	$n_{1\bullet}$
なし	c	d	$n_{2\bullet}$
塩基による小計	$n_{\bullet 1}$	$n_{\bullet 2}$	n_*

2. 全ゲノム関連解析

近年の実験技術の進歩によって，生命の情報に関する大規模な観測が可能となっている．特に，SNP の解析である GWAS では各被験者に対して数十万から一千万近い SNP 位置における変異を観測して，どの変異が疾患に関連しているかを調べる仮説検定が行われる．ここで被験者は観測，変異は説明変数，発症の有無は目的変数に対応し，GWAS では目的変数に統計的に有意に関連している説明変数をすべて求める．

GWAS で特によく調べられるのが，変異の有無と特定の疾患を発症しているか否かの関連である．N 人の被験者から変異の有無と発症の情報を得て，変異がある群とない群に分ける．また，発症の状態に関しても発症あり，なしの 2 群に分ける．このとき，変異と発症の関係は表 1 のような分割表として表される．

2.1 フィッシャー正確検定

フィッシャー正確検定 [9] は，2 つ以上のカテゴリーの独立性の検定を行う手法で，すべての可能な場合の列挙に基づく計算方法である．説明変数と目的変数の関係が表 1 のような 2×2 の表として表される場合には，p 値は周辺分布と呼ばれる表 1 の $n_{1\bullet}, n_{2\bullet}, n_{\bullet 1}, n_{\bullet 2}$ を固定した状態で，より偏りのある分布が現れる確率の総和として計算される．周辺分布を固定したときに分布 (a,b,c,d) が現れる確率 p_* は

$$p_* = \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{a!b!c!d!(a+b+c+d)!}$$

により計算される．周辺分布を固定したときの各分布は $i \in [-\min(a,d), \min(b,c)]$ として $(a+i, b-i, c-i, d+i)$ で与えられ，その確率 p_i は

$$p_i = \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{(a+i)!(b-i)!(c-i)!(d+i)!(a+b+c+d)!}$$

となる．

フィッシャー正確検定は近似を用いる他の手法よりも正確な確率を求められる一方で，多くの乗除を必要とし，計算に時間を要してしまうことが問題である．

2.1.1 表参照を使った実装

フィッシャー正確検定は階乗計算を含むため，標本数 N が大きい場合には直接計算するのが困難である．これを回避する現実的な解法として，分布の現れる確率そのも

のの代わりに確率の対数を計算する方法が知られている．$f(x) := \log(x!)$ とすると，確率 p_i の対数 $\log p_i$ は

$$\begin{aligned}\log p_i =& f(a+b) + f(c+d) + f(a+c) + f(b+d)\\ &- f(a+i) - f(b-i) - f(c-i) - f(d+i)\\ &- f(a+b+c+d)\end{aligned}$$

となる．あらかじめ $f(0), f(1), \cdots, f(N)$ を計算した表を用意し，$f(x)$ が必要となった場合には表を参照して $f(x)$ を読み出すことで，効率良く $\log p_i$ を計算することができる．表の計算は $f(x) = f(x-1) + \log x$ の関係式を使うことで容易に計算できる．

サンプル数の上界を N，有意水準を α とすると，フィッシャー正確検定はアルゴリズム 1 のように書ける．

3. 準備

3.1 記法，定義

本稿で扱う値はすべて有限体 $\mathbb{F}_p$ 上の値とする．ベクトル $\boldsymbol{a}$ の第 i 要素を $\boldsymbol{a}[i]$ で参照する．

3.1.1 計算効率の尺度

本稿で提案するアルゴリズムの効率はサブルーチンとして使用する各演算の効率に依存して決まる．本稿では秘密計算がマルチパーティプロトコルとして構成されるものとして計算効率の評価を行う．マルチパーティプロトコルは複数のパーティ間で通信を行いながら協調計算を行う方式である．一般的な構成では，各パーティが単独で行うローカルの計算に比べて通信に要する時間が著しく大きい．このため，ローカルの計算は無視できるものとみなし，通信したデータの量 (通信量) と一度に送ることのできるデータ量に制限がない場合に要する通信回数 (ラウンド数) の 2 つの尺度で計算効率の評価を行う．

3.2 既存の秘密計算上の演算

本稿で提案する秘密計算フィッシャー正確検定アルゴリズムは，既存の秘密計算上の演算の組み合わせにより構成する．本稿で用いる各演算は，semi-honest な攻撃者に対して情報理論的に安全に以下で述べる機能を実現する．

3.2.1 秘匿化，復元

$a \in \mathbb{F}_p$ を暗号化や秘密分散などの手段で秘匿化した値を a の秘匿文と呼び，$[\![a]\!]$ と表記する．また，a を $[\![a]\!]$ の平文と呼ぶ．ベクトル $\boldsymbol{v} = (\boldsymbol{v}[0], \ldots, \boldsymbol{v}[n-1])$ の各要素を秘匿化したベクトル $([\![\boldsymbol{v}[0]]\!], \ldots, [\![\boldsymbol{v}[n-1]]\!])$ を $[\![\boldsymbol{v}]\!]$ と表記する．本稿では Ben-Or らによる秘匿化，復元 [4] を用いる．有限体 $\mathbb{F}_p$ の大きさおよびパーティ数を定数とみなすと，秘匿化と復元は通信量 $O(1)$，ラウンド数 $O(1)$ である．

3.2.2 加算，減算，乗算

加算，減算，乗算の各演算は 2 つの値 $a, b \in \mathbb{F}_p$ の秘匿文 $[\![a]\!], [\![b]\!]$ を入力とし，それぞれ $a+b$，$a-b$，ab の計算結果 c_1，c_2，c_3 の秘匿文 $[\![c_1]\!]$，$[\![c_2]\!]$，$[\![c_3]\!]$ を計算する．これらの演算の実行をそれぞれ，

$$\begin{aligned}[\![c]\!] &\leftarrow \mathsf{Add}([\![a]\!], [\![b]\!]),\\ [\![c]\!] &\leftarrow \mathsf{Sub}([\![a]\!], [\![b]\!]),\\ [\![c]\!] &\leftarrow \mathsf{Mul}([\![a]\!], [\![b]\!])\end{aligned}$$

と記述する．誤解を招く恐れのない場合は，$\mathsf{Add}([\![a]\!], [\![b]\!])$，$\mathsf{Sub}([\![a]\!], [\![b]\!])$，$\mathsf{Mul}([\![a]\!], [\![b]\!])$ をそれぞれ $[\![a]\!] + [\![b]\!]$，$[\![a]\!] - [\![b]\!]$，$[\![a]\!] \times [\![b]\!]$ と略記する．本稿では Ben-Or らによる加算，減算 [4] および Gennaro らによる乗算 [11] を用いる．加算，減算は通信量およびラウンド数はともに 0 である．有限体 $\mathbb{F}_p$ の大きさおよびパーティ数を定数とみなすと，乗算は通信量 $O(1)$，ラウンド数 $O(1)$ である．

3.2.3 等号判定

等号判定の演算は 2 つの値 $a, b \in \mathbb{F}_p$ の秘匿文 $[\![a]\!], [\![b]\!]$ を入力とし，$a = b$ の真偽値 c の秘匿文 $[\![c]\!]$ を計算する．真偽値は真のとき 1，偽のとき 0 とする．この演算の実行を

$$[\![c]\!] \leftarrow ([\![a]\!] \stackrel{?}{=} [\![b]\!])$$

と記述する．Nishide と Ohta [19] により，$\mathbb{F}_p$ の大きさを ℓ とすると，通信量 $O(\ell)$，ラウンド数 $O(1)$ の手法が提案されている．

3.2.4 秘匿シフト

秘匿シフトの演算は大きさ n のベクトル $\boldsymbol{a} \in \mathbb{F}_p^n$ の秘匿文 $[\![\boldsymbol{a}]\!]$ と位置を表す値 $x \in \mathbb{F}_p$ の秘匿文を入力とし，大きさ n の $\boldsymbol{b}[i] = \boldsymbol{a}[x + i \mod n]$ $(0 \le i < n)$ を満たすベクトル $\boldsymbol{b} \in \mathbb{F}_p^n$ の秘匿文 $[\![\boldsymbol{b}]\!]$ を計算する．この演算の実行を

$$[\![\boldsymbol{b}]\!] \leftarrow \mathsf{PrivateLeftShift}([\![\boldsymbol{a}]\!]; [\![x]\!])$$

と記述する．濱田ら [22] により，通信量 $O(n)$，ラウンド数 $O(1)$ の手法が提案されている．

3.2.5 浮動小数点演算

小数点数 u を $\mathbb{F}_p$ の要素の組で表現することを考える．例えば，小数点数 u は $u = (1-2s)(1-z)v2^p$ $(z = 1 \text{ iff } u = 0,\ s = 1 \text{ iff } u > 0)$ として 4 つ組 (v, p, z, s) で表すことができる．このような u を表す $\mathbb{F}_p$ の要素の秘匿文の組を $\langle u \rangle$ と記述する．加算の演算は，2 つの値 a, b の秘匿文 $\langle a \rangle, \langle b \rangle$ を入力とし，$a+b$ の秘匿文 $\langle c_1 \rangle$ を計算する．減算の演算は，2 つの値 a, b の秘匿文 $\langle a \rangle, \langle b \rangle$ を入力とし，$a-b$ の秘匿文 $\langle c_2 \rangle$ を計算する．比較の演算は，2 つの値 a, b の秘匿文 $\langle a \rangle, \langle b \rangle$ を入力とし，$a < b$ の真偽値の秘匿文 $[\![c_3]\!]$ を計算する．自然指数関数の演算は，a の秘匿文 $\langle a \rangle$ を入力とし，e^a の秘匿文 $\langle c_4 \rangle$ を計算する．これらの演算の実行を

$$\begin{aligned}\langle c_1 \rangle &\leftarrow \mathsf{FLAdd}(\langle a \rangle, \langle b \rangle),\\ \langle c_2 \rangle &\leftarrow \mathsf{FLSub}(\langle a \rangle, \langle b \rangle),\\ [\![c_3]\!] &\leftarrow \mathsf{FLLT}(\langle a \rangle, \langle b \rangle),\\ \langle c_4 \rangle &\leftarrow \mathsf{FLExp}(\langle a \rangle)\end{aligned}$$

と記述する．誤解を招く恐れのない場合は，$\mathsf{FLAdd}(\langle a \rangle, \langle b \rangle)$，$\mathsf{FLSub}(\langle a \rangle, \langle b \rangle)$，をそれぞれ $\langle a \rangle + \langle b \rangle$，$\langle a \rangle - \langle b \rangle$ と略記する．

Algorithm 1 フィッシャー正確検定

入力: 2×2 の分割表の頻度 (a, b, c, d)，有意水準 α.
出力: 帰無仮説が棄却されるかどうか.
(1) $f(0) := 0, N := a+b+c+d$ とする．$x = 1$ から N について，$f(x) := f(x-1) + \log x$ を計算する.
(2) $-\min(a,d) \le i \le \min(b,c)$ の各 i に対して，p_i の対数を $\log p_i := f(a+b) + f(c+d) + f(a+c) + f(b+d) - f(a+b+c+d) - f(a+i) - f(b-i) - f(c-i) - f(d+i)$ により計算する.
(3) $\log p_i$ の自然指数関数により p_i を計算する.
(4) v を $p_i \le p_0$ を満たす p_i の総和とする.
(5) $v < \alpha$ ならば帰無仮説を棄却する.

Aliasgari らによって, 小数点数 u を $u = (1-2s)(1-z)v2^p$ として $\langle u \rangle := (\llbracket v \rrbracket, \llbracket p \rrbracket, \llbracket z \rrbracket, \llbracket s \rrbracket)$ とし，加算，比較，自然指数関数の各演算が統計的安全性の下で実現されている [2]. [2] で使われている統計的安全なサブプロトコルを情報理論的安全なサブプロトコルに置き換えることにより，$\mathbb{F}_p$ の大きさを ℓ とすると，加算が $O(\log \ell)$ ラウンド，$O(\ell \log \ell)$ 回の乗算で，比較が $O(\log \ell)$ ラウンド，$O(\ell \log \ell)$ 回の乗算で，自然指数関数が $O(\log^2 \ell)$ ラウンド，$O(\ell^2 \log \ell)$ 回の乗算で，それぞれ実現できる.

4. 秘密計算フィッシャー正確検定

2 節で見たように，フィッシャー正確検定は 2 種類のカテゴリー分けの間に関係があるかどうかを確かめるために使われる．GWAS では，注目しているカテゴリー分けと関係がある SNP がどれなのかを調べるために，すべての SNP に対して網羅的にこの検定を適用する．SNP の数は一般的に数百万程度と多いため検定は効率的に行われることが必要である．しかしながら 2.1.1 節で見たように標本数が大きい場合は計算に階乗が含まれるため，フィッシャー正確検定を参照表を使わずに効率的に実装するのは難しい．そこで，我々は表からの読み込みを効率良く行うアルゴリズムを作り，これを使って参照表を使ったフィッシャー正確検定に基づいた秘密計算用のアルゴリズムを作成する.

4.1 概要

提案する秘密計算フィッシャー正確検定アルゴリズムの概要は以下の通りであり，基本的には 2.1.1 節の表参照を用いるアルゴリズムの各演算を秘密計算の対応する演算に置き換えて作られる.

(1) $h+1$ 個 $(h = \lfloor N/2 \rfloor)$ の秘匿化された分布 $(\llbracket a_j \rrbracket, \llbracket b_j \rrbracket, \llbracket c_j \rrbracket, \llbracket d_j \rrbracket)$ を入力の分布 $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket, \llbracket d \rrbracket)$ から計算する.
(2) 関数 $\log(x!)$ を入力と $h+1$ 個の分布の各頻度に適用する.
(3) 頻度の階乗の対数を使って，入力と $h+1$ 個の分布それぞれの確率の対数を計算する.
(4) 入力と $h+1$ 個の分布それぞれについて計算した確率の対数に対して自然指数関数を適用し，各分布の確率を得る.
(5) 入力の分布の確率以下のすべての確率を足しあわせ，その値を p 値とする.
(6) p 値が有意水準より小さいかどうかを判定して出力する.

Algorithm 2 配列からの単一要素の読み込み

Notation: $\llbracket b \rrbracket \leftarrow \mathsf{PrivateRead}(\llbracket \boldsymbol{v} \rrbracket, \llbracket x \rrbracket)$.
Input: 大きさ n の配列の秘匿文 $\llbracket \boldsymbol{v} \rrbracket$，読み込み対象の要素の番号 $x \in \mathbb{F}_p$ の秘匿文 $\llbracket x \rrbracket$.
Output: $\llbracket b \rrbracket$ s.t. $b = \boldsymbol{v}[x]$.
1: **return** $\sum_{i=0}^{n} \llbracket \boldsymbol{v}[i] \rrbracket \times (\llbracket x \rrbracket \stackrel{?}{=} i)$.

Algorithm 3 配列からの一括読み込み

Notation: $\llbracket \boldsymbol{b} \rrbracket \leftarrow \mathsf{PrivateRangeRead}(\llbracket \boldsymbol{a} \rrbracket; \llbracket x \rrbracket; m)$.
Input: 大きさ n の配列の秘匿文 $\llbracket \boldsymbol{a} \rrbracket$，読み込み対象の開始位置の番号 $x \in \mathbb{F}_p$ の秘匿文 $\llbracket x \rrbracket$，読み込み範囲の大きさ $m \in \mathbb{F}_p$.
Output: 大きさ m の配列の秘匿文 $\llbracket \boldsymbol{b} \rrbracket$．ただし，各 $0 \le i < m$ について，$\boldsymbol{b}[i] = \boldsymbol{a}[x+i \mod n]$.
1: $\llbracket \boldsymbol{c} \rrbracket \leftarrow \mathsf{PrivateLeftShift}(\llbracket \boldsymbol{a} \rrbracket; \llbracket x \rrbracket)$.
2: **return** $(\llbracket \boldsymbol{c}[0 \mod n] \rrbracket, \ldots, \llbracket \boldsymbol{c}[m-1 \mod n] \rrbracket)$.

フィッシャー正確検定の p 値の計算では，周辺分布を固定したときの有効なすべての分布に対して確率を計算する必要がある．元の分布を (a, b, c, d) とすると，有効な分布の数は $\min(b,c) + \min(a,d) + 1$ 個となる．秘密計算で実現する場合，この数は明かさないまま計算しなくてはならない．そのため，この数の上界を x として，ダミーの分布を含めて常に x 個の分布を作成して計算を行う．$a+b+c+d \le N$ より $\min(b,c) + \min(a,d) \le N/2$ であるので，$h = \lfloor N/2 \rfloor$ とすると有効な分布の数は高々 $h+1$ である.

4.2 一括読み込みアルゴリズム

本節では秘密計算フィッシャー正確検定の部品となる，配列からの複数要素の読み込みを効率的に行うアルゴリズムを提案する．提案するアルゴリズムは，n 個の秘匿文の配列 $\llbracket \boldsymbol{a} \rrbracket$ と読み込み開始位置 x の秘匿文 $\llbracket d \rrbracket$，読み込む要素の数 m を入力とし，$\llbracket \boldsymbol{a} \rrbracket$ の x 番目から $x+m-1$ 番目までの要素を並べた秘匿文の配列を出力する.

まず，秘匿文の配列 $\llbracket \boldsymbol{a} \rrbracket$ から，x を明かさずに x 番目の要素だけ読み込む場合を考える．これはアルゴリズム 2 に示す方法により，n 要素の配列から $O(n)$ の通信量で実現することができる．これを m 回繰り返し実行することで通信量 $O(mn)$ で m 要素を読み込むことができるが，効率が悪い.

提案アルゴリズムは，まず配列 $[\![\boldsymbol{a}]\!]$ を左に x だけシフトし，その後シフト後の配列の先頭から m 要素を出力することで，$O(n)$ の通信量で m 要素の読み込みを実現する．このアルゴリズムをアルゴリズム 3 に示す．

4.3 提案アルゴリズム

本稿で提案する秘密計算フィッシャー正確検定アルゴリズムをアルゴリズム 5 に示す．このアルゴリズムは基本的には 2.1.1 節の表参照を利用したアルゴリズムの各演算を秘密計算の演算に置き換えることにより構成されているが，大きく 2 つの点で異なる．

まず 1 つめは，確率の計算式に階乗が含まれており計算が困難であるため，直接計算する代わりにテーブル参照に置き換えている点である．テーブル参照にはアルゴリズム 2 に示す単純な読み込みアルゴリズムを使う方法もあるが，アルゴリズム 2 は通信量が配列サイズの線形であり，$O(N)$ 回のテーブル参照を行うと通信量が $\Omega(N^2)$ になってしまって効率が悪い．この問題に対処するため，分布ごとの確率計算を行う際に，頻度は分布ごとに 1 ずつしか違わないことに着目する．すなわち，$p_0, \ldots, p_h$ を計算する場合に $a_i = a_{i-1} + 1$ の関係が成り立っていることから，各 a_i を個別にテーブル参照で $f(a_i)$ を読み込む代わりに一括読み込みアルゴリズム (アルゴリズム 3) で一度に $f(a_0), \ldots, f(a_h)$ を読み込む．これにより表参照は 9 回の従来の読み込みアルゴリズムの実行と 4 回の一括読み込みアルゴリズムにより実現でき，この部分の通信量を $O(N)$ とすることができる．

2 つめは，周辺分布を満たす分布の数を秘匿する必要があることである．フィッシャー正確検定では周辺分布を満たすすべての分布に対する確率を計算しなくてはならないが，その際に分布の数を隠す必要がある．前節で見た通り分布の数の上界は $h+1$ であるので，分布を順序付け，分布のうち最も小さいものを起点として $h+1$ 個の分布を生成する．まず ステップ 8 で $[\![a]\!]$ を動かすときの起点を求める．そして，動かした $[\![a]\!], [\![b]\!], [\![c]\!], [\![d]\!]$ を使ってそれぞれの確率を計算する．注意するべきは，$[\![a]\!], [\![b]\!], [\![c]\!], [\![d]\!]$ のいずれかが 0 未満になる場合である．これに対しては，ステップ 9 で表参照を行うときに対処する．すなわち，頻度 x が 0 未満の場合にはその階乗の対数 $f(x)$ を十分に小さい値 $-E$ とする．こうすることで，0 未満の頻度が含まれる場合には結果的に確率が十分に大きくなり，そのような分布はステップ 17 で確率の総和を求める際に除外されることになる．

4.4 計算量

提案アルゴリズムは $N+h+1$ 回の小数点数の分散，1 回の比較，2 回の条件付き割り当て，4 回の $N+h+1$ の配列からの一括読み込み，9 回の $N+h+1$ の配列からの読み込み，$h+2$ 回の 9 要素の浮動小数点総和，$h+2$ 回の浮動小数点自然指数関数，$h+2$ 回の浮動小数点数比較，$h+1$ 要素の浮動小数点総和からなる．3.2 節で述べた各演算を用いて可能な限り並列に実行したとすると，パーティ数と体 $\mathbb{F}_p$ の大きさを定数として，通信量が $O(N)$，ラウンド数が $O(\log N)$ である．

4.5 安全性

提案アルゴリズムは安全な部分アルゴリズムの組み合わせにより構成されている．アルゴリズム中でどの秘密の値も復元されることはないため，提案アルゴリズムも安全である．

5. おわりに

本稿では，フィッシャー正確検定を計算する秘密計算用のアルゴリズムを提案した．標本数の上界を N として，体の大きさとパーティ数を定数とみなすとき，素朴なアルゴリズムは通信量が $\Omega(N)$ となるのに対し，提案アルゴリズムは通信量が $O(N)$ である．

提案アルゴリズムの効率的な通信量は秘匿文の配列から位置を明かさずに複数の要素を取り出す通信量が配列サイズの線形の一括読み込みアルゴリズムによる．従来は n 要素の配列から m 要素を読み込む $O((m+n)\log(m+n))$ のアルゴリズムが提案されていたが，本稿では m 個の要素の相対位置が公開できる場合に限定し，$O(n)$ の通信量で m 要素を読み込めるアルゴリズムを提案した．

参考文献

[1] Aggarwal, G., Mishra, N. and Pinkas, B.: Secure Computation of the k th-Ranked Element, *EUROCRYPT*, pp. 40–55 (2004).
[2] Aliasgari, M., Blanton, M., Zhang, Y. and Steele, A.: Secure Computation on Floating Point Numbers, *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*, The Internet Society (2013).
[3] Ben-David, A., Nisan, N. and Pinkas, B.: FairplayMP: a system for secure multi-party computation, *ACM Conference on Computer and Communications Security* (Ning, P., Syverson, P. F. and Jha, S., eds.), ACM, pp. 257–266 (2008).
[4] Ben-Or, M., Goldwasser, S. and Wigderson, A.: Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract), *STOC* (Simon, J., ed.), ACM, pp. 1–10 (1988).
[5] Bogdanov, D., Laur, S. and Willemson, J.: Sharemind: A Framework for Fast Privacy-Preserving Computations, *ESORICS* (Jajodia, S. and López, J., eds.), LNCS, Vol. 5283, Springer, pp. 192–206 (2008).
[6] Burkhart, M., Strasser, M., Many, D. and Dimitropoulos, X. A.: SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics, *USENIX Security Symposium*, USENIX Association, pp. 223–240 (2010).
[7] Chida, K., Morohashi, G., Fuji, H., Magata, F., Fujimura, A., Hamada, K., Ikarashi, D. and Yamamoto, R.: Implementation and evaluation of an efficient secure

Algorithm 4 条件付き代入

Notation: $[\![b]\!] \leftarrow \mathsf{IfElse}([\![c]\!], [\![a_1]\!], [\![a_0]\!])$.
Input: ビットの秘匿文 $[\![c]\!]$, $a_1, a_0 \in \mathbb{F}_p$ の秘匿文 $[\![a_1]\!]$, $[\![a_0]\!]$.
Output: $[\![a_1]\!]$ if $c = 1$, $[\![a_0]\!]$ if $c = 0$.
1: **return** $[\![c]\!] \times ([\![a_1]\!] - [\![a_0]\!]) + [\![a_0]\!]$.

Algorithm 5 秘密計算フィッシャー正確検定

Input: 2×2 の分割表の度数の秘匿文 $([\![a]\!], [\![b]\!], [\![c]\!], [\![d]\!])$，標本数の上界 N，有意水準 α.
Output: 頻度 (a, b, c, d) に対する帰無仮説が有意水準 α で棄却された場合は $[\![1]\!]$，そうでない場合は $[\![0]\!]$.
Require: $a + b + c + d \leq N$.
1: $h \leftarrow \lfloor N/2 \rfloor$.
2: $[\![n_{1\bullet}]\!] \leftarrow [\![a]\!] + [\![b]\!]$.
3: $[\![n_{2\bullet}]\!] \leftarrow [\![c]\!] + [\![d]\!]$.
4: $[\![n_{\bullet 1}]\!] \leftarrow [\![a]\!] + [\![c]\!]$.
5: $[\![n_{\bullet 2}]\!] \leftarrow [\![b]\!] + [\![d]\!]$.
6: $[\![n_*]\!] \leftarrow [\![n_{1\bullet}]\!] + [\![n_{2\bullet}]\!]$.
7: E を大きい値，$f(x) := \begin{cases} \log(x!) & \text{if } 0 \leq x \leq N, \\ -E & \text{otherwise} \end{cases}$ として，$(f(0), f(1), \ldots, f(N+h))$ を要素ごとに秘匿して $\langle \boldsymbol{f} \rangle$ とする．
8: $[\![e]\!] \leftarrow \mathsf{IfElse}([\![a]\!] < [\![d]\!], [\![a]\!], [\![d]\!])$.
9: $(\langle a'_0 \rangle, \langle a'_1 \rangle, \ldots, \langle a'_h \rangle) \leftarrow \mathsf{PrivateRangeRead}(\langle \boldsymbol{f} \rangle; [\![a]\!] - [\![e]\!]; h+1)$.
10: $(\langle b'_h \rangle, \langle b'_{h-1} \rangle, \ldots, \langle b'_0 \rangle) \leftarrow \mathsf{PrivateRangeRead}(\langle \boldsymbol{f} \rangle; [\![b]\!] + [\![e]\!] - h; h+1)$.
11: $(\langle c'_h \rangle, \langle c'_{h-1} \rangle, \ldots, \langle c'_0 \rangle) \leftarrow \mathsf{PrivateRangeRead}(\langle \boldsymbol{f} \rangle; [\![c]\!] + [\![e]\!] - h; h+1)$.
12: $(\langle d'_0 \rangle, \langle d'_1 \rangle, \ldots, \langle d'_h \rangle) \leftarrow \mathsf{PrivateRangeRead}(\langle \boldsymbol{f} \rangle; [\![d]\!] - [\![e]\!]; h+1)$.
13: $\mathsf{PrivateRead}(\langle \boldsymbol{f} \rangle; \cdot)$ を $[\![a]\!], [\![b]\!], [\![c]\!], [\![d]\!], [\![n_{1\bullet}]\!], [\![n_{2\bullet}]\!], [\![n_{\bullet 1}]\!], [\![n_{\bullet 2}]\!], [\![n_*]\!]$ にそれぞれ適用し，
その結果を $\langle a'_{-1} \rangle, \langle b'_{-1} \rangle, \langle c'_{-1} \rangle, \langle d'_{-1} \rangle, \langle n'_{1\bullet} \rangle, \langle n'_{2\bullet} \rangle, \langle n'_{\bullet 1} \rangle, \langle n'_{\bullet 2} \rangle, \langle n'_* \rangle$ とする．
14: **for** $i = -1$ **to** h **do**
15: $\langle p_i \rangle \leftarrow \mathsf{FLExp}(\langle n'_{1\bullet} \rangle + \langle n'_{2\bullet} \rangle + \langle n'_{\bullet 1} \rangle + \langle n'_{\bullet 2} \rangle - \langle n'_* \rangle - \langle a'_i \rangle - \langle b'_i \rangle - \langle c'_i \rangle - \langle d'_i \rangle)$.
16: **end for**
17: $\langle v \rangle \leftarrow \sum_{i=0}^{h} \mathsf{IfElse}(\mathsf{FLLT}(\langle p_{-1} \rangle, \langle p_i \rangle), 0, \langle p_i \rangle)$.
18: **return** $\mathsf{FLLT}(\langle v \rangle, \alpha)$.

computation system using 'R' for healthcare statistics, *Journal of the American Medical Informatics Association*, Vol. 21, No. e2, pp. e326–e331 (2014).
[8] Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J. B. and Toft, T.: Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation, *TCC*, pp. 285–304 (2006).
[9] Fisher, R. A.: On the interpretation of χ 2 from contingency tables, and the calculation of P, *Journal of the Royal Statistical Society*, Vol. 85, No. 1, pp. 87–94 (1922).
[10] Geisler, M.: Cryptographic Protocols: Theory and Implementation, PhD Thesis, University of Aarhus (2010).
[11] Gennaro, R., Rabin, M. O. and Rabin, T.: Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography, *PODC* (Coan, B. A. and Afek, Y., eds.), ACM, pp. 101–111 (1998).
[12] Goldreich, O., Micali, S. and Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority, *STOC*, ACM, pp. 218–229 (1987).
[13] Goldreich, O. and Ostrovsky, R.: Software Protection and Simulation on Oblivious RAMs, *J. ACM*, Vol. 43, No. 3, pp. 431–473 (online), DOI: 10.1145/233551.233553 (1996).
[14] Goodrich, M. T.: Randomized Shellsort: A Simple Oblivious Sorting Algorithm, *SODA*, pp. 1262–1277 (2010).
[15] Henecka, W., Kögl, S., Sadeghi, A.-R., Schneider, T. and Wehrenberg, I.: TASTY: tool for automating secure two-party computations, *ACM Conference on Computer and Communications Security* (Al-Shaer, E., Keromytis, A. D. and Shmatikov, V., eds.), ACM, pp. 451–462 (2010).
[16] Kamm, L., Bogdanov, D., Laur, S. and Vilo, J.: A new way to protect privacy in large-scale genome-wide association studies, *Bioinformatics*, Vol. 29, No. 7, pp. 886–893 (online), DOI: 10.1093/bioinformatics/btt066 (2013).
[17] Lu, W.-J., Yamada, Y. and Sakuma, J.: Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption, *BMC medical informatics and decision making*, Vol. 15, No. Suppl 5, p. S1 (2015).
[18] Ning, C. and Xu, Q.: Multiparty Computation for Modulo Reduction without Bit-Decomposition and a Generalization to Bit-Decomposition, *ASIACRYPT*, pp. 483–500 (2010).
[19] Nishide, T. and Ohta, K.: Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol, *PKC*, pp. 343–360 (2007).
[20] Yao, A. C.-C.: How to Generate and Exchange Secrets (Extended Abstract), *FOCS*, pp. 162–167 (1986).
[21] Yates, F.: Contingency tables involving small numbers and the χ 2 test, *Supplement to the Journal of the Royal Statistical Society*, Vol. 1, No. 2, pp. 217–235 (1934).
[22] 濱田浩気，桐淵直人，五十嵐大：ラウンド効率のよい秘密計算パターンマッチング，コンピュータセキュリティシンポジウム 2014 論文集，Vol. 2014, No. 2, pp. 674–681 (2014).