Computer Security Symposium 2020
26 - 29 October 2020

秘密計算によるプライバシー保護生存時間解析

三品 気吹[1,a)]　深見 匠[1]　濱田 浩気[1]　五十嵐 大[1]　菊池 亮[1]

概要：秘密計算はデータを暗号化したま計算する技術である．そのため，プライバシーを保護したまま安全にデータ分析を行う方法として注目されている．特に医療分野のデータは高いプライバシーが要求されるため，様々な医療統計分析を秘密計算で行う研究がなされてきた．本稿では医療統計などで行われる「生存時間解析」のうち一般化 Wilcoxon 検定と cox 比例ハザード回帰という 2 つの分析手法を秘密計算上で実現する．生存時間解析は秘密計算での計算コストが大きい Group-by sum を何度も行い，更に cox 比例ハザード回帰は指数関数や除算なども含むため，秘密計算で高速に処理することは容易ではない．本稿ではそれらの処理コストが大きい計算を行う回数を最小限に抑えた秘密計算一般化 Wilcoxon 検定と秘密計算 cox 比例ハザード回帰を提案する．秘密計算一般化 Wilcoxon 検定は 10 万件のデータを約 0.2 秒で処理し，計算結果は平文と一致した．また秘密計算 cox 比例ハザード回帰は 8 属性 686 件の実データを 2.6 秒で処理し，計算結果は平文と一致した．

キーワード：秘密計算，医療統計，生存時間解析

Privacy Preserving Survival Analysis in Secure Computation

Ibuki Mishina[1,a)]　Takumi Fukami[1]　Koki Hamada[1]　Dai Ikarashi[1]　Ryo Kikuchi[1]

Abstract: Secure Computation makes it possible to perform various data analyses such as statistics and machine learning in a secure manner, since the data is still encrypted. Since data in the medical field requires a high level of privacy, a number of studies have been done to perform various medical statistical analyses in secure computation. In this paper, we implement two methods of survival analysis, the generalized Wilcoxon test and cox proportional hazards regression, both of which are used in medical statistics and other analyses in secure computation. Survival analysis is not easy to process in secure computation because the group-by-sum calculation is very expensive, and the Cox proportional hazards regression includes exponential functions and division. In this paper, we propose a secure generalized Wilcoxon test and a secure Cox proportional hazards regression that minimize the number of calculations of high processing costs. The secure generalized Wilcoxon test processed 100,000 data in about 0.2 seconds, and the results agreed with the plaintext. The secure Cox proportional hazards regression processed 686 real data of 8 attributes in 2.6 seconds, and the calculation results were in agreement with the plaintext.

Keywords: Secure Computation, Medical Statistics, Survival Analysys

1. はじめに

秘密計算は，データを暗号化したまま計算する技術である．秘密計算を用いることで，企業の重要な情報や顧客情報を安全に守ったまま，データ分析などに利用できるため，これまで様々な統計手法や機械学習手法を秘密計算で行うといった研究がなされてきた．本稿では医療分野で頻繁に用いられ，分析に高いプライバシーの安全性が要求される生存時間解析を取り扱う．生存時間解析には様々な手法があるが，本稿では一般化 Wilcoxon 検定と cox 比例ハザード回帰の 2 つを秘密計算上で行うアルゴリズムを提案し，実装・評価する．

1.1 関連研究

筆者らはこれまでにも，ログランク検定やフィッシャー正確確率検定といった医療統計の著名な手法を秘密計算で実現する取り組みをしている [9]．また医療統計や他の領

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories
[a)] ibuki.mishina.br@hco.ntt.co.jp

域でもよく用いられる手法であるロジスティック回帰についても，高速・高精度に秘密計算上で実現している [5][8].

1.2 本稿の貢献

本稿の貢献は大きく以下の 2 点である．

- 高速・高精度な秘密計算 Wilcoxon 検定アルゴリズムの提案・実装
- 高速・高精度な秘密計算 cox 比例ハザード回帰アルゴリズムの提案・実装

生存時間解析は，時点ごとに死亡したか生存していたか，というような集計を行う Group-by sum を頻繁に行うため秘密計算で高速に処理するのは難しい．また cox 比例ハザード回帰では，指数，除算といった秘密計算の苦手な実数演算も多く含む．

本稿では Group-by sum と指数・除算などの実数演算を最小限の回数で一般化 Wilcoxon 検定と cox 比例ハザード回帰を行うアルゴリズムを初めて実現した．そして，提案手法に対して大規模なダミーデータを用いた処理性能の評価と，実データを用いた精度の評価を実施し，提案手法の高い実用性を示した．

2. 準備

2.1 記法

a を b で定義することを $a := b$ と書き，ベクトルを $\vec{a} := (a_0, \ldots, a_{n-1})$ と書き，特筆しない限り A のような大文字は行列を表し，その転置行列は $A^\top$ と書く．

加減乗算において入力がベクトル $\vec{a}$ もしくは行列 A とスカラー b の場合，$\vec{a}, A$ の全ての要素に対して b との演算を行うものとする．また，特に記載が無いベクトルは列ベクトルである．横ベクトルの場合は ${}^t\vec{a}$ のように左上に t を付けることで区別する．

2.2 秘密計算

2.2.1 秘密分散を用いた秘密計算

秘密計算にはいくつか方式があり，中でも秘密情報を「シェア」という複数の断片に変換する秘密分散方式は，データの処理単位が小さく，処理が高速である [2], [11]. 本稿では，n 個のシェアを生成し，k 個以上のシェアからは秘密が復元できるが，k 個未満のシェアからは秘密の情報が全く漏れない (k, n) 閾値法という秘密分散方式を用いる．平文とシェア (暗号文) を区別するため，平文 a の暗号文は $[\![a]\!]$ と書き，括弧がついていないものは平文とする．

2.2.2 算術演算

四則演算

2 つの暗号文 $[\![a]\!], [\![b]\!]$ の加算，減算，乗算は，それぞれ暗号文 $[\![a+b]\!], [\![a-b]\!], [\![a\times b]\!]$ を計算する処理である．これらの演算をそれぞれ，$[\![a]\!]+[\![b]\!], [\![a]\!]-[\![b]\!], [\![a]\!]\times[\![b]\!]$ と書く，また，暗号文 $[\![a]\!]$ を平文 b で割る処理は $[\![a]\!]/b$ のような記法とする．入力がベクトルや行列で，要素ごとにこれらの処理を行う場合も同様に $[\![\vec{a}]\!]+[\![\vec{b}]\!], [\![A]\!]+[\![B]\!]$ のような記法とする．

加減乗算において入力が行列 A と列ベクトル $\vec{b}$ の場合は，行列の各列ベクトルに対して $\vec{b}$ との要素ごとの演算を実施し，行列 A と行ベクトル ${}^t\vec{b}$ 場合は，行列の各行ベクトルに対して ${}^t\vec{b}$ との要素ごとの演算を実施するものとする．

総和

ベクトル $[\![\vec{a}]\!]$ の総和を求める処理を $\mathsf{sum}([\![\vec{a}]\!])$ と記述する．また $\mathsf{sum}([\![A]\!])$ のように $m\times n$ の行列が入力の場合は，列方向の総和を計算し，長さ n のベクトル $[\![\vec{c}]\!]$ を出力するものとする．

積和

長さが等しい 2 つのベクトル $[\![\vec{a}]\!], [\![\vec{b}]\!]$ の積和を求める処理を $\mathsf{psum}([\![\vec{a}]\!], [\![\vec{b}]\!])$ と記述する．また大きさの等しい $m\times n$ 行列 $[\![A]\!], [\![B]\!]$ を入力した場合，$\mathsf{psum}([\![A]\!], [\![B]\!])$ は列方向の積和を計算し，長さ n のベクトル $[\![\vec{c}]\!]$ を出力するものとする．行列 $[\![A]\!], [\![B]\!]$ の行方向の積和を計算する場合は，$\mathsf{hpsum}([\![A]\!], [\![B]\!])$ と書き，$\mathsf{hpsum}([\![A]\!], [\![B]\!])$ は長さ m のベクトル $[\![\vec{c}]\!]$ を出力する．

prefix sub

ベクトル $[\![\vec{a}]\!] := ([\![a_1]\!], [\![a_2]\!], \cdots, [\![a_n]\!])$ とスカラー $[\![b]\!]$ から $([\![b]\!], [\![b]\!]-[\![a_1]\!], [\![b]\!]-([\![a_1]\!]+[\![a_2]\!]), \cdots, [\![b]\!]-\sum[\![\vec{a}]\!])$ となるベクトルを計算することを，$\mathsf{prefixSub}([\![\vec{a}]\!], [\![b]\!])$ と書く

2.2.3 比較演算

等号

2 つの暗号文 $[\![a]\!], [\![b]\!]$ もしくは暗号文 $[\![a]\!]$ と平文 b が等しいかどうかを判定する処理を $[\![c]\!] \leftarrow \mathsf{equality}([\![a]\!], [\![b]\!])$, $[\![\vec{c}]\!] \leftarrow \mathsf{equality}([\![a]\!], b)$ と書く．出力は 1 か 0 の暗号文である．入力がベクトルの場合も同じ記法とする．

2.2.4 実数演算

逆数

暗号文 $[\![a]\!]$ の逆数 $1/[\![a]\!]$ を計算することを $[\![c]\!] \leftarrow \mathsf{reciprocal}([\![a]\!])$ のように書く．入力がベクトルの場合も同じ記法とする．

指数

暗号文 $[\![a]\!]$ を入力とし，ネイピア数 e の $[\![a]\!]$ 乗を計算することを $[\![c]\!] \leftarrow \mathsf{exp}([\![a]\!])$ のように書く．入力がベクトルの場合も同じ記法とする．

2.2.5 ベクトル・行列の変形操作

行列 $\leftrightarrow$ ベクトルの変形

行列 $[\![A]\!]$ の列ベクトルを 1 列目から順番に繋げたベクトルに変形することを $[\![\vec{a}]\!] \leftarrow \mathsf{flatten}([\![A]\!])$ と書き，ベクトル $[\![\vec{a}]\!]$ を列数 n の行列に変形することを $[\![A]\!] \leftarrow \mathsf{matrixify}([\![\vec{a}]\!], n)$ と書く．matrixify は flatten の逆操作であり，ベクトル $[\![\vec{a}]\!]$ を n 等分したベクトルに切り分け，それを順番に列ベクトルとして並べる．

転置

ベクトル形式の行列 $[\![\vec{a}]\!]$ の転置行列を求める処理を，

$[\![\vec{c}]\!] \leftarrow \mathsf{transpose}([\![\vec{a}]\!], n)$ と書く．n は $[\![\vec{a}]\!]$ の列数である．$[\![\vec{c}]\!]$ を列数 (つまり $[\![\vec{a}]\!]$ の行数) で matrixify すると $[\![A^\top]\!]$ となる．

ベクトルの複製操作

ベクトル $[\![\vec{a}]\!] := ([\![a_1]\!], [\![a_2]\!], \cdots, [\![a_n]\!])$ の各要素が順番に b 個ずつ並んだベクトル $([\![a_1]\!], \cdots, [\![a_1]\!], \cdots, [\![a_n]\!], \cdots, [\![a_n]\!])$ を求める処理を $[\![\vec{c}]\!] \leftarrow \mathsf{copyVec}([\![\vec{a}]\!], b)$ と書く．

また，ベクトル $[\![\vec{a}]\!] := ([\![a_1]\!], [\![a_2]\!], \cdots, [\![a_n]\!])$ を b 回繰り返したベクトル $([\![a_1, \cdots, a_n, \cdots, a_1, \cdots, a_n]\!])$ を求める処理を $[\![\vec{c}]\!] \leftarrow \mathsf{repeatVec}([\![\vec{a}]\!], b)$ と書く．

2.2.6 Group-by common

Group-by common は, Group-by sum や Group-by count といった様々な Group-by 演算で共通的に用いることができる中間データを生成する処理である [7]．中間データは置換表 $[\![\vec{\pi}]\!]$ と，キーの値の境目かどうかを表すフラグ $[\![\vec{e}]\!]$ を含み，これらを使いまわすことで，同じキーを用いた様々な Group-by 演算を効率良く行う．

キーのベクトル $[\![\vec{k}]\!]$ を入力して Group-by common を行うことを式 (1), $[\![\vec{\pi}]\!]$ を用いてベクトル $[\![\vec{a}]\!]$ や行列 $[\![A]\!]$(行数が $[\![\vec{\pi}]\!]$ の長さと等しい) をソートする処理を式 (2)(3)，ソート済みのベクトル $[\![\vec{a'}]\!]$ や $[\![\vec{A'}]\!]$ と $[\![\vec{e}]\!]$ を用いて Group-by sum を行うことを式 (4)(5) のように記述する．$[\![\vec{a'}]\!], [\![A']\!]$ などのプライムはソート済みであることを表し，以降も同様の記法とする．

$$[\![\vec{\pi}]\!], [\![\vec{e}]\!] \leftarrow \mathsf{groupByCommon}([\![\vec{k}]\!]) \quad (1)$$

$$[\![\vec{a'}]\!] \leftarrow \mathsf{sort}([\![\vec{a}]\!], [\![\vec{\pi}]\!]) \quad (2)$$

$$[\![A']\!] \leftarrow \mathsf{sort}([\![A]\!], [\![\vec{\pi}]\!]) \quad (3)$$

$$[\![\vec{c}]\!] \leftarrow \mathsf{groupBySum}([\![\vec{a'}]\!], [\![\vec{e}]\!]) \quad (4)$$

$$[\![C]\!] \leftarrow \mathsf{groupBySum}([\![A']\!], [\![\vec{e}]\!]) \quad (5)$$

sort, groupBySum の入力が行列の場合，処理は列ごとに行われる．また一般的に Group-by sum を行うと出力のサイズは入力のサイズ以下になるが，本稿では入力と出力のサイズは同じであり，不要な分は末尾が 0 でパディングされているものとする．

2.2.7 その他の処理

要素がすべて $[\![a]\!]$ で長さ b のベクトルを作成する処理を $[\![\vec{c}]\!] \leftarrow \mathsf{fill}([\![a]\!], b)$ と書く．また $\mathsf{ifgate}(a, b, c)$ は $a = 1$ の場合は b，$a = 0$ の場合は c を出力する処理である．

2.2.8 プログラマブルな秘密計算ライブラリ MEVAL

MEVAL は筆者らが開発する秘密分散ベースの秘密計算ライブラリで，前述したような演算を組み合わせて自由にプログラムできる [13]．本稿の実験で用いたプログラムは，MEVAL を用いて実装している．

2.3 生存時間解析

生存時間解析とは「イベントが発生するまでの時間」に着目した統計学の一領域である．ここでの「イベント」とは，患者の死亡や機械の故障，ユーザが利用中のサービスを解約する，といったものが挙げられ，幅広い分野で応用されている．たとえば医療分野であれば，患者の健康状態や年齢，性別，投与した薬の種類等を特徴量として，生存時間 (患者が生存している期間) との相関を分析し，何年後に生存している確率は何%といった予測を行ったり，複数の試薬の効果を比較する目的などで行われる [12].

2.3.1 一般化 Wilcoxon 検定

一般化 Wilconxon 検定は 2 つの群の生存時間を比較するための検定手法であり, 試薬や臨床試験の効果測定などに用いられる．この検定は，データがある分布に従っているという前提のないノンパラメトリックな検定手法である．

表 1 生存表の例

時刻 t	状態 s	群 g
2	1	A
8	0	A
4	0	B
20	1	B
21	1	B

表 2 のように生存時間 t と 2 値の状態 $s \in \{0, 1\}$, 2 つの群 $g \in \{A, B\}$ を属性にもつとする．例えば, 状態と群は $s \in \{0 =$ 打ち切り, $1 =$ 死亡 $\}$, $g \in \{0 =$ プラセボ, $1 =$ 実薬 $\}$ に対応する．このとき, カプランマイヤー法により, 各時刻における群ごとの生存率を表す生存曲線と呼ばれるグラフが得られる．この生存曲線は視覚的にわかりやすいが, 統計的な差の有無はわかりづらい．一般化 Wilcoxon 検定は生存曲線の統計的な差の有無を判別する検定である．

一般化 Wilcoxon 検定では, 両群の生存曲線が等しいという帰無仮説を立て, それが棄却されるかどうかを計算する．具体的な計算手順を下記に示す．

(1) 状態 $s = 1$ の各時刻毎に表 2 のようなクロス表を作成する．生存数とは時刻 t_i の直前において死亡も打ち切りも発生していない標本数であり, 死亡数とは時刻 t_i において新たに死亡した標本数である．

(2) クロス表を集計し, 各時刻毎の下記の値を計算する．

i) A 群の生存数 n_i^A，B 群の生存数 n_i^B．

ii) 全体の生存数 $n_i = n_i^A + n_i^B$．

iii) A 群の死亡数 o_i^A，B 群の死亡数 n_i^B．

iv) 全体の死亡数 $o_i = o_i^A + o_i^B$．

v) A 群の死亡数期待値 $e_i^A = n_i^A \cdot \frac{o_i}{n_i}$．

一般化 Wilcoxon 検定では, 時刻により死亡数の重みを変更する．重みはその時刻の両群合わせた生存数 n_i を用いる．

(3) A 群の死亡数と死亡数期待値の差 $u = \sum_{i=0}^{k-1}(o_i^A - e_i^A)n_i$ を計算する．

(4) u の分散 $V = \sum_{i=0}^{k-1} \frac{n_i^A n_i^B o_i (n_i - o_i)}{(n_i - 1)}$ を計算する．

(5) u^2/v と自由度 1 の χ^2 分布から p 値を計算する．

(6) p 値をもとに帰無仮説を棄却するか（生存曲線に差があるか）を判断する．

表 2 $t = t_i$ の時のクロス表

時刻 $t = t_i$	死亡数	生存数	合計
群 A	o_i^A	n_i^A	$o_i^A + n_i^A$
群 B	o_i^B	n_i^B	$o_i^B + n_i^B$
合計	o_i	n_i	$o_i + n_i$

類似の手法としてログランク検定があるが，ログランク検定では全ての時点での死亡を平等に扱うのに対し，一般化 Wilcoxon 検定では時間経過につれて重みを小さくしていくという違いがある．一般化 Wilcoxon 検定でこのような重みの調節を行うのは「後ろの時点ほどデータが減っているため信頼性が低い」という考えに基づくものである．ただ，どちらの検定手法のほうが総合的に優れているということはなく，状況に応じて選択する．

2.3.2 cox 比例ハザード回帰

一般化 Wilcoxon 検定は，「プラセボを投与した群 A」と「実薬を投与した群 B」というように，「実薬を投与したか否か」という 1 つの特徴量に着目して比較する手法であった．それに対し cox 比例ハザード回帰では，年齢，性別，喫煙習慣，治療法など様々な特徴量を用いて生存時間との関連を分析し，cox 比例ハザードモデルを作る手法である．一般化 Wilcoxon 検定で有意な差の見られた特徴量のいくつかを，cox 比例ハザードモデルに組み込むといった使い方などが考えられる．cox 比例ハザードモデルは目的変数が生か死かといった 2 値であるという点でロジスティック回帰モデルに似ており，大まかなイメージとしてはロジスティック回帰に時間の要素が加わったものである．

cox 比例ハザードモデル

cox 比例ハザードモデルは式 (6) のような形をとる [3].

$$\lambda(t|\vec{z}) = \lambda_0(t)\exp(\vec{\beta}^\top \vec{z}) \tag{6}$$

t, β, z はそれぞれ時間, 重み, 特徴量を表し，$\lambda_0(t), \exp(\vec{\beta}^\top \vec{z})$ はそれぞれベースラインハザード関数，相対危険度関数 (ハザード) と呼ばれる．相対危険度関数を見ると，重回帰モデルやロジスティック回帰モデルと同じ線形結合モデルであることが分かる．cox 比例ハザード回帰では，重回帰分析やロジスティック回帰分析と同様に，線形結合モデルの重みを推定をする．重みの推定方法としては cox 本人によって提唱された部分尤度法を用いる [3].

部分尤度

cox 比例ハザードモデルにおける部分尤度は式 (7) のようになる．詳細な導出が必要であれば cox の提案論文 [3] を参考にされたい．

$$L(\vec{\beta}) = \prod_{i=1}^{D}\left[\frac{\exp(\vec{\beta}^\top \vec{z}_i)}{\sum_{j\in R_i}\exp(\vec{\beta}^\top \vec{z}_j)}\right] \tag{7}$$

D は死亡が観測された時点の数であり，$\vec{z}_i$ は時点 i に死亡した患者の特徴量を表す．また R_i は時点 i の直前まで打ち切りも死亡も発生していない患者の集合であり，リスクセットと呼ばれる．従って式 (7) の部分尤度関数は，時点毎に (死亡した患者のハザード)/(リスクセットのハザードの総和) を計算し，全時点分掛け合わせたものである．この部分尤度は同じ時刻に複数の打ち切りや死亡が発生していない (タイデータが無い) という仮定を置いているため，タイデータがあることの多い実データでは，次に示す Breslow 法がよく用いられる．

Breslow 法

Breslow の部分尤度関数を式 (8) に示す．

$$L(\vec{\beta}) = \prod_{i=1}^{D}\left[\frac{\exp(\vec{\beta}^\top \vec{s}_i)}{\{\sum_{j\in R_i}\exp(\vec{\beta}^\top \vec{z}_j)\}^{d_i}}\right] \tag{8}$$

基本的には式 (7) の cox の部分尤度ど同じであるが，分母が d_i 乗 (d_i は時点 i の死亡患者数) されている点と，分子に $\vec{z}_i$ の代わりに，時点 i の死亡患者の特徴量の総和 s_i を用いる点である．$d_i = 1$ の場合は cox の部分尤度と一致する．

式 (8) の尤度関数を最大化する $\vec{\beta}$ の最尤推定量を求める方法としては，Newton 法などが一般的である．Newton 法では，式 (8) を対数尤度関数へと変形したのち，その対数尤度関数の 1 階微分 (勾配) と 2 階微分 (ヘシアン) を用いて計算する．対数尤度関数 $l(\vec{\beta})$ と，その 1 階微分 $U(\vec{\beta})$ および 2 階微分 $I(\vec{\beta})$ を式 (9)〜式 (11) に示す

$$l(\vec{\beta}) = \sum_{i=1}^{D}\left\{\vec{\beta}^\top \vec{s}_i - d_i \log\sum_{j\in R_i}\exp(\vec{\beta}^\top \vec{z}_j)\right\} \tag{9}$$

$$U(\vec{\beta}) = \sum_{i=1}^{D}\left[\vec{s}_i - \frac{d_i\sum_{j\in R_i}\vec{z}_j\exp(\vec{\beta}^\top \vec{z}_j)}{\sum_{j\in R_i}\exp(\vec{\beta}^\top \vec{z}_j)}\right] \tag{10}$$

$$I(\vec{\beta}) = \sum_{i=1}^{D} d_i\left[\frac{\sum_{j\in R_i}\vec{z}_j\vec{z}_j{}^\top\exp(\vec{\beta}^\top \vec{z}_j)}{\sum_{j\in R_i}\exp(\vec{\beta}^\top \vec{z}_j)} - \frac{\{\sum_{j\in R_i}\vec{z}_j\exp(\vec{\beta}^\top \vec{z}_j)\}\{\sum_{j\in R_i}\vec{z}_j\exp(\vec{\beta}^\top \vec{z}_j)\}^\top}{\{\sum_{j\in R_i}\exp(\vec{\beta}^\top \vec{z}_j)\}^2}\right] \tag{11}$$

Newton 法では，式 (10) と式 (11) を用いて，以下の式 (12) を反復して $\vec{\beta}$ の最尤推定値を求める．およそ 5 回ほどの反復で収束する．

$$\vec{\beta} = \vec{\beta} + I(\vec{\beta})^{-1}U(\vec{\beta}) \tag{12}$$

2.4 共役勾配法

式 (12) からも分かるように，Newton 法ではヘシアンの逆行列の計算が必要になる．逆行列の計算はコストが大きいため，回避する方法として共役勾配法が知られている．共役勾配法では $I(\vec{\beta})^{-1}$ を計算せずに，$I(\vec{\beta})$ と $U(\vec{\beta})$ から直接 $I(\vec{\beta})^{-1}U(\vec{\beta})$ を求める手法である．共役勾配法のアル

ゴリズムを **Algorithm** 1 に示す ϵ は収束判定のハイパーパラメータを表し，0^n は 0 が n 個並んだベクトルを表す．

Algorithm 1 共役勾配法

Require: n 次の正定値対称行列 H，n 次のベクトル $\vec{g}$
Ensure: $H\vec{d} = \vec{g}$ となる n 次のベクトル $\vec{d}$

$\vec{d} \leftarrow 0^n$
$\vec{r} \leftarrow \vec{g}$
$\vec{p} \leftarrow \vec{g}$
$\rho \leftarrow \vec{r}^\top \vec{r}$
while $\rho > \epsilon$ **do**
  $\alpha \leftarrow \frac{\vec{r}^\top \vec{p}}{\vec{p}^\top H \vec{p}}$
  $\vec{d} \leftarrow \vec{d} + \alpha \vec{p}$
  $\vec{r} \leftarrow \vec{r} - \alpha H \vec{p}$
  $\beta \leftarrow \frac{\vec{r}^\top \vec{r}}{\rho}$
  $\vec{p} \leftarrow \vec{r} + \beta \vec{p}$
  $\rho \leftarrow \vec{r}^\top \vec{r}$
end while

3. 提案手法

3.1 固定小数点数によるアルゴリズム設計

秘密計算では浮動小数点数の処理コストが大きいため，本稿では固定小数点数を用いてアルゴリズム設計を行う．固定小数点数の計算では乗算などによって小数点位置が変わってしまうため，適宜右シフトによって小数点位置を調節している．本稿の実装では，定数ラウンドの高速な右シフト [10] を採用している．アルゴリズムが煩雑になるのを防ぐため，以降に示すアルゴリズム上では右シフトを行う箇所を明記しない．

3.2 Group-by common を用いた効率的なアルゴリズム

一般化 Wilcoxon 検定や cox 比例ハザード回帰では Group-by sum を処理の中で何度も行うが，全てキーが同一であるため，最初に一度だけ行った Group-by common で得た $[\![\vec{\pi}]\!]$ と $[\![\vec{e}]\!]$ を使いまわすことができる．提案手法では Group-by common を活用し，秘密計算一般化 Wilcoxon 検定と秘密計算 cox 比例ハザード回帰を効率良く計算する．

3.3 秘密計算一般化 Wilcoxon 検定

本稿では, 秘密計算一般化 Wilcoxon 検定の入力は, 時刻 t，状態 $s \in \{0, 1\}$，群 $g \in \{A, B\}$ とし, 出力は 2.3.1 における u, v とする．u, v を取得した者が平文値で以降の計算をし, 検定結果を得る．

Algorithm 2 はソートされた各時刻 t_i における各群の生存数 $n_i^{\{A,B\}}$ を計算する．

Algorithm 3 はソートされた各時刻 t_i における各群の死亡数 $o_i^{\{A,B\}}$ を計算する．

Algorithm 4 に提案する秘密 Wilcoxon 検定プロトコルを示す．countSurvivor および countDead は，ぞれぞれ

Algorithm 2 Count Survivor

Require: m 次の群 A, B のデータ位置ベクトル $[\![\vec{g^A}]\!], [\![\vec{g^B}]\!]$，$m$ 次フラグベクトル $[\![\vec{e}]\!]$
Ensure: 各群の時刻ごとの生存数ベクトル $[\![\vec{n^A}]\!], [\![\vec{n^B}]\!]$

$[\![\vec{s^A}]\!] \leftarrow \mathsf{groupBySum}([\![\vec{g^A}]\!], [\![\vec{e}]\!])$
$[\![\vec{s^B}]\!] \leftarrow \mathsf{groupBySum}([\![\vec{g^B}]\!], [\![\vec{e}]\!])$
$[\![\vec{n_0^A}]\!] \leftarrow \mathsf{sum}([\![\vec{s^A}]\!])$
$[\![\vec{n_0^B}]\!] \leftarrow \mathsf{sum}([\![\vec{s^B}]\!])$
for $i = 1$ to $n - 1$ **do**
  $[\![n_i^A]\!] \leftarrow [\![n_{i-1}^A]\!] - [\![s_{i-1}^A]\!]$
  $[\![n_i^B]\!] \leftarrow [\![n_{i-1}^B]\!] - [\![s_{i-1}^B]\!]$
end for

Algorithm 3 Count Dead

Require: m 次の状態ベクトル $[\![\vec{s}]\!]$，m 次の群 A, B のデータ位置ベクトル $[\![\vec{g^A}]\!], [\![\vec{g^B}]\!]$, m 次フラグベクトル $[\![\vec{e}]\!]$.
Ensure: 各群の時刻ごとの死亡数ベクトル $[\![\vec{o^A}]\!], [\![\vec{o^B}]\!]$.

$[\![\vec{d^A}]\!] \leftarrow [\![\vec{s}]\!] \times [\![\vec{g^A}]\!]$
$[\![\vec{d^B}]\!] \leftarrow [\![\vec{s}]\!] \times [\![\vec{g^B}]\!]$
$[\![\vec{o^A}]\!] \leftarrow \mathsf{groupBySum}([\![\vec{d^A}]\!], [\![\vec{e}]\!])$
$[\![\vec{o^B}]\!] \leftarrow \mathsf{groupBySum}([\![\vec{d^B}]\!], [\![\vec{e}]\!])$

Algorithm 4 Secure Wilcoxon Test

Require: m 次の時刻ベクトル $[\![\vec{t}]\!]$，m 次の状態ベクトル $[\![\vec{s}]\!]$，m 次の群ベクトル $[\![\vec{g}]\!]$.
Ensure: $[\![u]\!], [\![v]\!]$

$[\![\vec{\pi}]\!], [\![\vec{e}]\!] \leftarrow \mathsf{groupByCommon}([\![\vec{t}]\!])$
$[\![\vec{g'}]\!] \leftarrow \mathsf{sort}([\![\vec{g}]\!], [\![\vec{\pi}]\!])$
$[\![\vec{g^A}]\!] \leftarrow \mathsf{equality}([\![\vec{g'}]\!], A)$
$[\![\vec{g^B}]\!] \leftarrow \mathsf{equality}([\![\vec{g'}]\!], B)$
$[\![\vec{n^A}]\!], [\![\vec{n^B}]\!] \leftarrow \mathsf{countSurvivor}([\![\vec{g^A}]\!], [\![\vec{g^B}]\!], [\![\vec{e}]\!])$
$[\![\vec{o^A}]\!], [\![\vec{o^B}]\!] \leftarrow \mathsf{countDead}([\![\vec{s}]\!], [\![\vec{g^A}]\!], [\![\vec{g^B}]\!], [\![\vec{e}]\!])$
$[\![\vec{n}]\!] \leftarrow [\![\vec{n^A}]\!] + [\![\vec{n^B}]\!]$
$[\![\vec{n'}]\!] \leftarrow [\![\vec{n}]\!] - 1$
$[\![\vec{o}]\!] \leftarrow [\![\vec{o^A}]\!] + [\![\vec{o^B}]\!]$
$[\![\vec{n'}_{inv}]\!] \leftarrow \mathsf{reciprocal}([\![\vec{n'}]\!], b_\alpha, b_\beta)$
$[\![cond]\!] \leftarrow \mathsf{equality}(\vec{n'}, [\![0]\!])$
$[\![\vec{n'}_{inv}]\!] \leftarrow \mathsf{ifgate}([\![cond]\!], [\![\vec{n'}]\!], [\![\vec{n'}_{inv}]\!])$
$[\![u]\!] \leftarrow \mathsf{sum}\Big([\![\vec{o_i^A}]\!] \times [\![\vec{n_i}]\!] - [\![\vec{n_i^A}]\!] \times [\![\vec{o_i}]\!]\Big)$
$[\![v]\!] \leftarrow \mathsf{sum}\Big([\![\vec{n_i^A}]\!] \times [\![\vec{n_i^B}]\!] \times [\![\vec{o_i}]\!] \times ([\![\vec{n_i}]\!] - [\![\vec{o_i}]\!]) \times [\![\vec{n'}_{iinv}]\!]\Big)$

Algorithm 2，Algorithm 2 に対応する．Algorithm 2, Algorithm 3 の入力となる群 A, B のデータ位置 $[\![\vec{g^A}]\!], [\![\vec{g^B}]\!]$ は Algorithm 4 の 1 から 3 行目で示す, 各データ群 A, B のどちらであるかという判定値である．このアルゴリズムでは, 入力値の秘匿化のため, 存在しない時刻での集計値も $n^{\{\vec{A},B\}}, o^{\{\vec{A},B\}}$ に含まれる．そのような時刻での統計量を 0 とするため, $[\![\vec{n'}_{inv}]\!]$ を計算する際に, $[\![\vec{n'}]\!] = [\![0]\!]$ ならば $[\![\vec{n'}_{inv}]\!] = [\![0]\!]$ となるよう計算している．

3.4 秘密計算 Cox 比例ハザード回帰

3.4.1 処理コストの大きい演算の削減

cox 比例ハザード回帰では式 (10)(11) を計算し，**Algorithm** 1 に示した共役勾配法を用いて式 (12) を反復計算することで，重み $\vec{\beta}$ の最尤推定値を求める．式 (10)(11) か

ら分かる通り，exp や除算が多く含まれ，また $\sum_{j \in R_i}$ の処理は Group-by sum であるため，秘密計算での計算コストが非常に大きい．

本稿では cox 比例ハザード回帰における exp，除算，Group-by sum といったコストの大きい処理を最小限に抑え，効率良く計算するアルゴリズムを提案する．単純に式 (10)(11) の通りに計算した場合，Newton 法の 1 反復あたり exp は 7 回，除算は 3 回 (共役勾配法の分は除く)，Group-by sum は 7 回必要になるが，本稿では下記の通り最小限に抑えた．

- exp の計算が 1 反復あたり 1 回
- 逆数の計算が 1 反復あたり 1 回
- Group-by common が処理全体で 1 回

この 3 点について，もう少し詳細を述べる．

exp の削減

exp の引数が全て $\vec{\beta}^\top \vec{z_j}$ あるため，一度計算したら後は使い回せば良い．

除算の削減

除算を逆数の計算＋乗算で行う場合，$\sum_{j \in R_i} \exp(\vec{\beta}^\top \vec{z_j})$ の逆数を，式 (10) の第 2 項と式 (11) の第 1 項で使い回せるため，除算 2 回ではなく逆数の計算 1 回＋乗算 2 回で済む．式 (11) の第 2 項は前述の 2 つとは除数が異なるが，この項は除算をしなくても求めることができる．式 (10) の第 2 項と式 (11) の第 2 項を見比べてみると，式 (10) の第 2 項の d_i を除いた部分を A としたとき，式 (11) の第 2 項は $AA^\top$ で表せるため，行列積のみで式 (11) の第 2 項は計算できる．

Group-by sum の削減

これについては先述の 3.2 で述べた通りである．各 $\sum_{j \in R_i}$ の計算では，境目を表すフラグ $[\![\vec{e}]\!]$ を用いた集計を行うのみである．

3.4.2 繰り返し処理の削減

式 (10) や式 (11) を式の通りに計算すると，$i = 1$ から $i = D$ の時点まで順番に計算していき，最後に sum を求めるのがシンプルな方法であるが，この方法では時点の数だけ繰り返し処理を行うことになる．インタープリタ型言語で効率よく計算するためには繰り返し処理を避けたほうが良いため，全時点分をまとめて一度に計算するアルゴリズムを提案する．

このように 1 レコードのベクトルではなく全レコード分の行列の状態で処理する際も，基本的には 1 レコードずつ処理する場合と大きくは変わらないが，cox 比例ハザード回帰において鬼門となるのは式 (11) の第 1 項に含まれる $\vec{z_j}\vec{z_j}^\top$ の計算と，第 2 項の計算である．全時点分をまとめて計算する場合，この 2 つは 3 階のテンソルになる．1 レコードずつの処理の場合，$\vec{z_j}\vec{z_j}^\top$ は列ベクトルと行ベクトルの掛け算であるため出力は行列になる．この処理を全レコードに対して行うため，行列がレコード数分並んだ 3 階のテンソルとなる．

このように 2 つの行列の列ベクトル同士を掛け算し，出力が 3 階のテンソルになる処理は少し複雑に見えるが，行列の形状操作を工夫することによってベクトル同士の乗算 1 回で実現できる．行列の形状操作はローカルでの処理であるため，秘密計算でも処理コストが大きくならない．

Algorithm 5 matmul 2d to 3d

Require: $m \times n$ の行列 $[\![A]\!]$，$[\![B]\!]$ を flatten した $[\![\vec{a}]\!], [\![\vec{b}]\!]$
Ensure: 長さ $m \times n \times n$ のベクトル $[\![\vec{c}]\!]$
  $[\![\vec{x_a}]\!] \leftarrow \mathsf{repeatVec}([\![\vec{a}]\!], n)$
  $[\![\vec{x_a}]\!] \leftarrow \mathsf{transpose}([\![\vec{x_a}]\!], n \times n)$
  $[\![\vec{x_b}]\!] \leftarrow \mathsf{transpose}([\![\vec{b}]\!], n)$
  $[\![\vec{x_b}]\!] \leftarrow \mathsf{copyVec}([\![\vec{x_b}]\!], n)$
  $[\![\vec{c}]\!] \leftarrow [\![\vec{x_a}]\!] \times [\![\vec{x_b}]\!]$
  $[\![\vec{c}]\!] \leftarrow \mathsf{transpose}([\![\vec{c}]\!], m)$

3.4.3 秘密計算 cox 比例ハザード回帰アルゴリズム

前述の工夫を取りいれた秘密計算 cox 比例ハザード回帰アルゴリズムの全体を **Algorith** 6 に示す．**Algorith** 6 に出てくる calcGH は勾配とヘシアンを計算する処理であり，処理が長いため別途 **Algorith** 7 に示した．また CG は共役勾配法を計算する処理であり，筆者らが [6] で提案したものであるため，本稿ではアルゴリズムの記載を割愛する．

Algorithm 6 秘密計算 cox 比例ハザード回帰

Require: $m \times n$ の特徴量行列 $[\![Z]\!]$，m 次の状態ベクトル $[\![\vec{c}]\!]$，
  m 次の時刻ベクトル $[\![\vec{t}]\!]$，学習回数 α
Ensure: n 次の重みベクトル $[\![\vec{\beta}]\!]$
  $[\![\vec{\beta}]\!] \leftarrow \mathsf{fill}([\![0]\!], n)$
  $[\![\vec{\pi}]\!], [\![\vec{e}]\!] \leftarrow \mathsf{groupByCommon}([\![\vec{t}]\!])$
  $[\![Z']\!] \leftarrow \mathsf{sort}([\![Z]\!], [\![\vec{\pi}]\!])$
  $[\![\vec{c'}]\!] \leftarrow \mathsf{sort}([\![\vec{c}]\!], [\![\vec{\pi}]\!])$
  時点毎の死亡例の特徴量の総和 $[\![S]\!]$ の計算
  $[\![Z'_{dead}]\!] \leftarrow [\![Z']\!] \times [\![\vec{c'}]\!]$
  $[\![S]\!] \leftarrow \mathsf{groupBySum}([\![Z'_{dead}]\!], [\![\vec{e}]\!])$
  時点毎の死亡数 $[\![d]\!]$ の計算
  $[\![\vec{d}]\!] \leftarrow \mathsf{groupBySum}([\![\vec{c'}]\!], [\![\vec{e}]\!])$
  $zz^\top$ の計算
  $[\![\vec{z'}]\!] \leftarrow \mathsf{flatten}([\![Z']\!])$
  $[\![zz']\!] \leftarrow \mathsf{matmul2to3}([\![\vec{z'}]\!], [\![\vec{z'}]\!])$
  Newton-CG 法による $[\![\vec{\beta}]\!]$ の推定
  for $i = 0, 1, \cdots, \alpha - 1$ **do**
    $[\![\vec{g}]\!], [\![H]\!] \leftarrow \mathsf{calcGH}([\![\vec{\beta}]\!], [\![\vec{z'}]\!], [\![S]\!], [\![zz']\!], [\![\vec{d}]\!])$
    $[\![\vec{p}]\!] \leftarrow \mathsf{CG}([\![\vec{g}]\!], [\![H]\!])$
    $[\![\vec{\beta}]\!] \leftarrow [\![\vec{\beta}]\!] + [\![\vec{p}]\!]$

$[\![zz']\!]$ は 3 階のテンソルである．

Algorithm 7 勾配とヘシアンの秘密計算

Require: n 次ベクトル $[\![\vec{\beta}]\!]$, $m \times n$ 行列 $[\![Z']\!]$, $m \times n$ 行列 $[\![S]\!]$, $m \times n \times n$ テンソル $[\![zz']\!]$,m 次ベクトル $[\![\vec{d}]\!]$
Ensure: n 次ベクトル $[\![\vec{g}]\!]$, $n \times n$ 行列 $[\![H]\!]$
$[\![\vec{u'}]\!] \leftarrow \mathsf{hpsum}([\![Z']\!], [\![\vec{\beta}]\!])$
$[\![\vec{v'}]\!] \leftarrow \mathsf{exp}([\![\vec{u'}]\!])$
$[\![\vec{v'}_{repeat}]\!] \leftarrow \mathsf{repeatVec}([\![\vec{v'}]\!], n)$
$[\![\vec{w'}]\!] \leftarrow [\![\vec{x'}]\!] \times [\![\vec{v'}_{repeat}]\!]$
$[\![\vec{v'}_{repeat}]\!] \leftarrow \mathsf{repeatVec}([\![\vec{v'}]\!], n \times n)$
$[\![\vec{x'}]\!] \leftarrow [\![zz']\!] \times [\![\vec{v'}_{repeat}]\!]$
$[\![W']\!] \leftarrow \mathsf{matrixify}([\![\vec{w'}]\!], n)$
$[\![X']\!] \leftarrow \mathsf{matrixify}([\![\vec{x'}]\!], n \times n)$
各時点の $\sum_{j \in R_i} \exp(\vec{\beta}^\top \vec{z_j})$ の計算
$[\![\vec{v}_{gsum}]\!] \leftarrow \mathsf{groupBySum}([\![\vec{v'}]\!], [\![\vec{e}]\!])$
$[\![v_{sum}]\!] \leftarrow \mathsf{sum}([\![\vec{v}_{gsum}]\!])$
$[\![\vec{v}_{psub}]\!] \leftarrow \mathsf{prefixSub}([\![\vec{v}_{gsum}]\!], [\![v_{sum}]\!])$
各時点の $\sum_{j \in R_i} \vec{z_j} \exp(\vec{\beta}^\top \vec{z_j})$ の計算
$[\![W_{gsum}]\!] \leftarrow \mathsf{groupBySum}([\![W']\!], [\![\vec{e}]\!])$
$[\![\vec{w'}_{sum}]\!] \leftarrow \mathsf{sum}([\![W_{gsum}]\!])$
$[\![W_{psub}]\!] \leftarrow \mathsf{prefixSub}([\![W_{gsum}]\!], [\![\vec{w}_{sum}]\!])$
各時点の $\sum_{j \in R_i} \vec{z_j} \vec{z_j}^\top \exp(\vec{\beta}^\top \vec{z_j})$ の計算
$[\![X_{gsum}]\!] \leftarrow \mathsf{groupBySum}([\![X']\!], [\![\vec{e}]\!])$
$[\![\vec{x}_{sum}]\!] \leftarrow \mathsf{sum}([\![X_{gsum}]\!])$
$[\![X_{psub}]\!] \leftarrow \mathsf{prefixSub}([\![X_{gsum}]\!], [\![\vec{x}_{sum}]\!])$
$\sum_{j \in R_i} \exp(\vec{\beta}^\top \vec{z_j})$ の逆数の計算
$[\![\vec{y}]\!] \leftarrow \mathsf{reciprocal}([\![\vec{v}_{psub}]\!])$
勾配の計算
$[\![\vec{y}_{repeat}]\!] \leftarrow \mathsf{repeatVec}([\![\vec{y}]\!], n)$
$[\![\vec{w}_{psub}]\!] \leftarrow \mathsf{flatten}([\![W_{psub}]\!])$
$[\![\vec{g}_{tmp}]\!] \leftarrow [\![\vec{w}_{psub}]\!] \times [\![\vec{y}_{repeat}]\!]$
$[\![\vec{d}_{repeat}]\!] \leftarrow \mathsf{repeatVec}([\![\vec{d}]\!], n)$
$[\![\vec{g}_{tmp2}]\!] \leftarrow [\![\vec{d}_{repeat}]\!] \times [\![\vec{g}_{tmp}]\!]$
$[\![\vec{s}]\!] \leftarrow \mathsf{flatten}([\![S]\!])$
$[\![\vec{g}_{tmp2}]\!] \leftarrow [\![\vec{s}]\!] - [\![\vec{g}_{tmp2}]\!]$
$[\![G]\!] \leftarrow \mathsf{matrixify}([\![\vec{g}_{tmp2}]\!], n)$
$[\![\vec{g}]\!] \leftarrow \mathsf{sum}([\![G]\!])$
ヘシアンの計算
$[\![\vec{y}_{repeat}]\!] \leftarrow \mathsf{repeatVec}([\![\vec{y}]\!], n \times n)$
$[\![\vec{x}_{psub}]\!] \leftarrow \mathsf{flatten}([\![X_{psub}]\!])$
$[\![\vec{h}_{tmp}]\!] \leftarrow [\![\vec{x}_{psub}]\!] \times [\![\vec{r}_{repeat}]\!]$
$[\![\vec{h}_{tmp2}]\!] \leftarrow \mathsf{matmul2to3}([\![\vec{g}_{tmp}]\!], [\![\vec{g}_{tmp}]\!])$
$[\![\vec{h}_{tmp2}]\!] \leftarrow [\![\vec{h}_{tmp}]\!] - [\![\vec{h}_{tmp2}]\!]$
$[\![H]\!] \leftarrow \mathsf{matrixify}([\![\vec{h}_{tmp2}]\!], n \times n)$
$[\![\vec{h}]\!] \leftarrow \mathsf{sum}([\![H]\!])$

4. 実験

4.1 実験設定

4.1.1 測定環境

表 3 に示すマシン 3 台を用いて実験を行った．

表 3 測定環境

OS	CentOS Linux release 7.3.1611
CPU	Intel Xeon Gold 6144k(3.50GHz 8 コア/16 スレッド) × 2
メモリ	768GB
NW	Intel Ethernet Controller X710/X557-AT 10G リング構成

4.1.2 データセット

データセットとして，R の Survival パッケージ等で提供されている German Breast Cancer Study Group(GBSG)[4] というデータセットを用いた．説明変数が 8 個 × データ数 686 件のデータセットである．提案手法の秘密計算 Wilcoxon 検定，秘密計算 cox 比例ハザード回帰の両方で，実データとして GBSG データを用いる．

4.2 秘密計算 Wilcoxon 検定

ダミーデータを用いて，1000 件〜1000 万件のデータに対して提案手法の秘密計算 Wilcoxon 検定を行い，処理時間と計算結果 (p 値) の評価を行ったものを表 4 に示す．計算結果の精度を評価するため，平文で同じデータを処理した場合の結果も記載した．平文での処理は Python の生存時間解析パッケージである lifelines[1] を用いた．

表 4 秘密計算 Wilcoxon 検定の処理時間と計算結果

データ件数	処理時間 [s]	結果 (提案手法)	結果 (平文)
1000 件	0.047	0.91638	0.91638
1 万件	0.063	0.45852	0.45852
10 万件	0.197	0.32428	0.32428
100 万件	2.013	0.37537	0.37604
1000 万件	30.123	0.67920	0.67930

表 4 より，10 万件の処理時間が約 0.2 秒と非常に高速であり，1000 万件の更に大きなデータの場合でも 30 秒程度で処理できることが分かる．また，計算結果も 10 万件までは平文で計算したものと一致しており，100 万件以上で多少の差は生じるものの小さな誤差となった．

また，提案手法の秘密計算 Wilcoxon 検定に対して実データ (GBSG) を入力した場合の処理時間と計算結果 (p 値)，および平文での計算結果を表 5 に記載する．GBSG データの中で用いたのは horTh という説明変数である．この説明変数はホルモン治療を行ったか否かの 2 値である．

表 5 実データでの処理時間と計算結果

	処理時間 [s]	結果 (提案手法)	結果 (平文)
GBSG	0.052	0.00383	0.00383

実データを用いた実験でも提案手法が高速・高精度であることを示した．

4.3 秘密計算 Cox 比例ハザード回帰

ダミーデータを用いて，提案手法の秘密計算 cox 比例ハザード回帰の処理時間を測定した結果を表 6 に示す．時刻を表すデータの bit 数はソートの処理時間に影響を与えるため，今回の実験では GBSG データに合わせて，全てのダミーデータで $14bit$ 固定とした．

表 6 秘密計算 cox 比例ハザード回帰の処理時間

データ数	説明変数の数	処理時間 [s]
1000	10	2.500
1000	100	58.969
1 万	100	421.665
10 万	10	46.953

説明変数の数が 10，データ数が 1000 のケースでは処理時間は約 2.5 秒と非常に高速である．途中で説明変数の数の 2 乗の大きさの処理が生じるため，10 と 100 での差は 10 倍以上になるが，前述の理由があるため想定通りである．説明変数の数が 100 の場合は 10 の場合と比較して処理時間は長くなるが，データ数が 1 万件の場合でも 7 分程度であるため実用的な範囲であるといえる．また，説明変数の数が 10 であればデータ件数が 10 万件という大きなデータの場合でも約 47 秒で高速に処理できることを示した．

提案手法に対して GBSG データを用いた実験を行い，処理時間と精度を評価したものを表 7 に示す．精度の評価方法は，秘密計算 cox 比例ハザード回帰によって得られたパラメータと，平文 (lifelines) で cox 比例ハザード回帰を行った場合に得られたパラメータの相関係数を求めることで行った．相関係数は 1 に近いほど 2 つのデータが近いことを示す指標である．

表 7 実データでの実験

処理時間 [s]	パラメータの相関係数
2.662	0.9999

実データでも約 2.7 秒という高い処理性能を示し，相関係数についてもほぼ 1 となり，高い精度を示した．実際に出力されたパラメータを表 8 に示す．

表 8 パラメータの比較

	提案手法	平文
w_1	-0.0949	-0.09
w_2	0.1102	0.11
w_3	0.1631	0.16
w_4	0.2729	0.27
w_5	-0.4524	-0.45
w_6	0.0256	0.03
w_7	-0.1617	-0.16
w_8	-0.1318	-0.13

lifelines ではパラメータが小数点第 2 位までしか表示されないため，それ以上の精度での評価はできないが少なくとも第 2 位までは完全に一致することが確認できた．

5. おわりに

本稿の貢献は大きく以下の 2 点である．

- 高速・高精度な秘密計算 Wilcoxon 検定アルゴリズムの提案・実装
- 高速・高精度な秘密計算 cox 比例ハザード回帰アルゴリズムの提案・実装

また本稿では，生存時間解析のような Group-by 演算を何度も行う処理を，Grou-by common を用いて効率的に処理する方法や，行列の形状操作を工夫することで複雑な行列の計算を効率良く行う方法を示した．このような工夫は，今回取り扱った生存時間解析の 2 手法以外でも利用できると考えられる．

5.1 今後の展望

cox 比例ハザード回帰で全レコードを同時に計算する場合，レコード数 m と説明変数の数 n に対して，途中で $m \times n \times n$ のテンソルが出現する．そのため n が 100 程度かつ m が 10 万といった非常に大きなデータセットでは，処理時間がかかり過ぎたり，そもそもメモリが足らず処理できないといった問題が生じてしまう．しかしメモリ不足を防ぐために時点毎にレコード単位で処理を反復すると，今度は反復処理に時間がかかり過ぎてしまうという問題がある．そのためレコード数や属性数が多すぎる場合は，機械学習におけるミニバッチ処理のように，全レコードを同時に処理せずに何個かの時点でまとめて処理をするといった方法も考えられる．説明変数の数やレコード数がどの程度を超えたら前述のような手法をとるのが良いのかといった検討含め，今後も引き続き更なる高速化・大規模化によって実用性の向上を目指す．

参考文献

[1] lifelines. https://lifelines.readthedocs.io/en/latest/.

[2] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *CCS*, pp. 805–817, 2016.

[3] David R Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, Vol. 34, No. 2, pp. 187–202, 1972.

[4] Gunter von Minckwitz, Guˆˆccˆˆ88nter Raab, Angelika Caputo, Martin Schuˆˆccˆˆ88tte, Joˆˆccˆˆ88rn Hilfrich, Jens U Blohmer, Bernd Gerber, Serban D Costa, Elisabeth Merkle, Holger Eidtmann, et al. Doxorubicin with cyclophosphamide followed by docetaxel every 21 days compared with doxorubicin and docetaxel every 14 days as preoperative treatment in operable breast cancer: the geparduo study of the german breast group. *Journal of Clinical Oncology*, Vol. 23, No. 12, pp. 2676–2685, 2005.

[5] 三品気吹, 濱田浩気, 五十嵐大. 秘密計算によるロジスティック回帰は本当に使えるか？ In *SCIS*, 2019.

[6] 三品気吹, 濱田浩気, 五十嵐大, 菊池亮. 秘密実数演算を用いた高速かつ高精度なロジスティック回帰とデータ標準化. In *CSS*, 2020.

[7] 菊池亮, 濱田浩気, 五十嵐大, 高橋元, 高橋克巳. 横断的動線分析を秘密計算でやってみよう. In *SCIS*, 2020.

[8] 濱田浩気, 五十嵐大, 三品気吹, 菊池亮. 秘密計算上の一括近似とそれを使った正確度の高いロジスティック回帰. In *CSS*, 2019.

[9] 市川敦謙, 須藤弘貴, 竹之内大地, 五十嵐大, 濱田浩気, 菊池亮. 大規模な医療統計に向けた実用的な秘密分析計算. In *CSS*, 2019.

[10] 五十嵐大. 秘密計算 ai の実装に向けた秘密実数演算群の設計と実装 -o(p) ビット通信量 o(1) ラウンドの実数向け右シフト-. In *CSS*, 2019.

[11] 五十嵐大, 濱田浩気, 菊池亮, 千田浩司. 超高速秘密計算ソートの設計と実装：秘密計算がスクリプト言語に並んだ日. In *CSS*, 2017.

[12] 中村剛. cox 比例ハザードモデル. 朝倉書店, 2018.

[13] 桐淵直人, 五十嵐大, 濱田浩気, 菊池亮. プログラマブルな秘密計算ライブラリ MEVAL3. *SCIS*, 2018.