Computer Security Symposium 2017
23 - 25 October 2017

大規模データを実用的な速度で処理可能な匿名化ライブラリの設計と実装評価

長谷川聡[1]　正木彰伍[1]　岡田莉奈[1]

概要：本研究は, 大規模なデータに対し, 匿名化処理を実用的な時間で実行完了できることを目的とする. 大規模なデータは, メモリに乗り切らない場合, ディスク I/O や分散処理化などが必要になり, 処理が遅くなってしまう. 効率よく処理するためには, 出来る限りオンメモリで処理することが課題となる. 本稿では, データを圧縮することで記憶容量を抑えオンメモリ化し, 圧縮による遅延を減らすよう並列化処理の実装を工夫することで, 効率よく匿名化処理を実行するライブラリを開発したことを報告する. 1 億レコード 100 属性 (100 億セル, およそ 60GB) のデータに対し, スパコンのような大規模演算装置でなく, 一般的なサーバを利用した場合でも, 4 属性の k-匿名化処理を 1 時間 42 分という実用的な時間で完了した.

キーワード：PWS, 匿名化, 大規模データ

Design and Evaluation of A Practically Efficient Anonymization Library for Large Scale Data

Satoshi Hasegawa[1]　Shogo Masaki[1]　Rina Okada[1]

Keywords: PWS,Anonymization,Large Scale Data

1. はじめに

近年の人工知能やデータマイニング技術の発達により, 購買履歴や位置データといった個人に紐づく情報 (パーソナルデータ) の二次利用が注目を浴びている. しかしながらパーソナルデータは, 個人特定可能な情報が含まれており, 安易に第三者へデータを提供するなどの二次利用を行うと, プライバシを侵害するリスク生じる恐れがある.

パーソナルデータ利活用に向け, 個人情報保護法が改正され, 2017 年 5 月 30 日に全面施工された. 改正個人情報保護法では, パーソナルデータを**匿名加工情報**と呼ばれる一定の基準を満たした情報に加工すれば, 収集時とは異なる目的の利用を可能にするよう規定された.

パーソナルデータを匿名加工情報にする加工技術の 1 つとして, 「匿名化」技術があり, 法改正以前から盛んに研究されている [3, 7–9, 12]. 匿名化は, データベース中に含まれる個人データを加工し, 個人特定を困難にすることでプライバシ保護する技術をいう. 匿名化したデータがどの程度プライバシを保護しているかを示す代表的な指標として, k-匿名性 [12] が提案されている. k-匿名性とは, 「データベース中に同じ準識別子 (**Q**uasi **I**dentifier) [*1] を持つレコードが少なくとも k 個以上存在する」ことでプライバシ保護の度合いを表す. k-匿名性は, k 人未満に個人を絞り切れないという直感的にわかりやすい指標であるゆえ, 広く用いられている. 他にも, l-多様性 [10] や, 差分プライバシ [2] といった, k-匿名性では測ることができないプライバシ保護度合いを測る指標も提案されており, これらの指標についても満たしていることが望ましい. しかし, 匿名加工情報の作成で参考となる個人情報保護委員会事務局レ

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories

[*1] 準識別子とは, それ単体では個人が特定できないが, 組み合わせることで個人が特定可能な属性のことをいう. 例えば, 年齢, 住所, 性別などがそれらに該当すると言われている.

© 2017 Information Processing Society of Japan

表 1 匿名加工情報作成の際に用いる加工技法の一覧 [15,16,20]

加工技法	説明
ソート	レコードを一定の規則で並び替える
シャッフル	レコードの並び順を (確率的に) 変更する
仮 ID 化	対象となる属性の ID を仮 ID に変更する
項目削除	対象となる属性の削除する
レコード削除	対象となるレコードを削除する
一般化	対象となる属性の値を上位概念に置き換えること
トップ (ボトム) コーディング	対象となる属性の数値に対し, 大きい値または小さい値をまとめること
サンプリング	レコードの一部を無作為に抽出する
ミクロアグリゲーション (グルーピング)	グループの代表的な記述に置き換えること
丸め	四捨五入などにより, データを丸めること
スワッピング	対象となる属性の値を相互に (確率的に) 入れ替えること
ノイズ付与	対象となる属性の値に誤差を付与すること
k-匿名化	k-匿名性を満たすようデータを加工する

ポート [16] や [19] では, これらのプライバシ保護指標まで明確には言及していない. 本研究では, まずは, k-匿名性にフォーカスして議論を進め, k-匿名性以外のプライバシ保護指標の検討は今後の課題とする.

匿名加工情報を作成するにあたり, [15,16,20] では, データの一般化やランダム化, 削除といった基本的な加工技法に加え, 先程述べた k-匿名性を満たすアルゴリズム (k-匿名化) によるデータ加工技法を組み合わせて利用することを推奨している (表 1 参照). 匿名加工情報を作成するにあたり, k-匿名化だけでなく, さきほど述べたような基本的な加工技法も組み合わせて利用できることが望ましい.

1.1 大規模なデータの匿名加工情報の作成

匿名加工情報の作成対象となるデータは, 購買履歴や移動履歴データといった履歴データや, ユーザ数が数百万から一億規模のデータが想定されてる.

大規模なデータを匿名化するにあたり, 単純に匿名化処理を計算機上で実装しては, メモリ不足もしくは実用的な時間で匿名化処理ができない問題が起きてしまう. この問題を解決するため, 大きく 3 つのアプローチが研究されている.

1 つ目は, 適宜ディスクから必要に応じて適宜メモリに展開し, 匿名化処理を行うアプローチである. 例えば, データベースから SQL でデータを取り出し, 適宜匿名化処理を行うのがそれにあたる [8]. このアプローチの場合, 必要な量のデータしかメモリ上に展開しないため, 大規模なデータも扱うことが可能である. ただし, 必要に応じて, データベースからの読み書き処理が生じるため, ディスクの読み書き時間がボトルネックとなる. k-匿名化アルゴリズムの多くは, 繰り返し処理が多いことから, ディスク読み書きが頻繁に起きることが想定され, I/O ボトルネックの影響により, 実行時間が膨大になる恐れがある.

2 つ目は, データをコンパクトに表現することで省メモリ化し, オンメモリで匿名化を実施するアプローチである [4]. このアプローチの場合, 匿名化処理中にディスクの読み書きが発生しないため, 効率良く匿名化処理を実行することが可能となる. ただし, データのコンパクト表現は, 高々定数倍の省メモリ効果であるため, データ規模によっては本アプローチは利用できない場合がある.

3 つ目は, 複数台のマシンを用意し, これらのマシンで互いに協調しながら匿名化を実施するアプローチである. このアプローチの場合, 1 台ではカバーできないメモリ量を扱うことができる. ただし, ネットワークでの協調処理や, マシンの故障時の対応, 複数台マシンに適したアルゴリズムの設計の必要がある. また, データ規模によっては, ネットワークの速度や協調動作の処理によるボトルネックで, かえって動作が遅くなることも考えられる.

1.2 貢献

本研究では, 匿名加工情報の作成に適した匿名化処理のアーキテクチャを検討し, 実装評価した結果を報告する. 本研究の貢献は以下の 3 つである.

- データ規模に対する匿名化処理アーキテクチャの検討.
- 大規模データに対する効率の良い k-匿名化アルゴリズムの提案.
- 汎用サーバでも大規模データを実用的な時間で処理が可能なことの確認.

匿名加工情報の作成にあたり, どの程度のデータ規模で, どのような匿名化処理を行うかにより, 効率良く処理可能な適切なアーキテクチャが異なってくる. 本研究では, 想定されうるデータ規模や, どういった匿名化処理が行われるかを調査し, 匿名加工処理を行う上で適したアーキテクチャの検討を行った.

また, 大規模なデータが加工対象となることを想定し, 大規模データに対する加工処理アルゴリズムの検討を行った. [15,16,20] に示されている加工技法のうち, k- 匿名化の処理時間の影響が大きいため, k-匿名化アルゴリズムを実用的な処理時間で終えれるよう高速化の工夫を行った. 具体的には, k-匿名化のアルゴリズムを並列化することで, 複数 CPU の力を使い実行速度の向上を測った. k-匿名化アルゴリズムは, 単純に並列化できないことから, 一部を独立な処理に変更することで, 並列処理化可能な形に修正する検討を行った.

検討したアーキテクチャおよび匿名化アルゴリズムを実装し, 汎用サーバを用いて実行時間の評価を行った. オープンソースの既存の高速な k-匿名化処理を行うツールと比べて, メモリ消費量がおよそ半分以下であるにも関わらず, 1000 万レコード 100 属性の人工データに対し, 1.5 倍高速に処理できたことを確認した. また 1 億レコード 100 属性の人工データに対しては, 既存のツールではメモリ使用量

© 2017 Information Processing Society of Japan

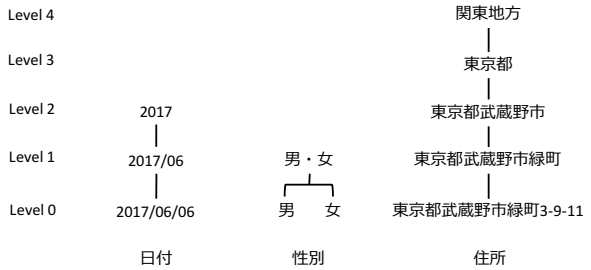

図 1 一般化階層の例. それぞれ日付, 性別, 住所の一般化階層を表す.

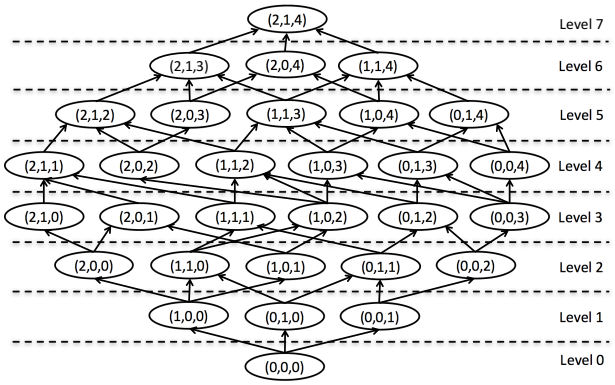

図 2 図 1 で示した日付 (高さ 3), 性別 (高さ 2), 住所 (高さ 5) の階層の一般化階層のラティス構造の例である. (0,0,0) はそれぞれレベル 0 の値であり, 例えば日付を 1 階層上げた場合, (1,0,0) となる. 図の右側に示すラティスのレベルは, 階層上げを何回行ったかを表している.

の関係上実行できなかったが, 本ライブラリでは実行を行うことができ, 1 時間 40 分程度という現実的な時間で終えることができた.

2. 準備

2.1 データ加工技法

匿名加工情報の加工技法として, [16] や [15,20] で示されているようなデータ加工技法 (ソート, シャッフル, 仮 ID 化, 項目削除, レコード削除, 一般化, トップ (ボトム) コーディング, サンプリング, ミクロアグリゲーション (グルーピング), 丸め, スワッピング, ノイズ付与) を組み合わせて適用することが考えられる. 加工技法の一覧およびその説明を表 1 に示す.

匿名加工情報を作成するプログラムを作成する場合, これらを具備して任意に使いまわせるような構成であることが望ましいといえる.

2.2 k-匿名化

k-匿名化は, データを一般化もしくは削除することにより, k-匿名性を満たすようにデータを加工するアルゴリズムのことを言う. 一般化処理は, 主に一般化階層と呼ばれる値の汎化・特化関係を表す木構造を準備し, その木構造を辿る (階層を上がる) ことで値のコーディングを行い処理をする. 図 1 に一般化階層の例を示す. 例えば, 図 1 に示す日付属性の一般化階層の場合, レベル 0 が現在の値 (2017/06/06), 1 階層上げたレベル 1 は年月単位 (2017/06), 2 階層上げたレベル 2 は年単位 (2017) となる.

k-匿名化の際に, ある属性の同じ値に対し同じ一般化レベルで処理を施すか (グローバルリコーディングという), 同じ属性の同じ値でも一般化レベルを変更して処理を施すか (ローカルリコーディング) で, それぞれ様々な手法が提案されている [8,9]. 本研究では, [11] も指摘の通り, 分析の際に同じ属性の一般化レベルが同じであるほうが利便性が良いことを考慮し, グローバルリコーディングによる k-匿名化に注目する.

k-匿名性を満たす最低限の一般化および削除処理を行うアルゴリズム (以下, 最適な k-匿名化アルゴリズムという) は, 各属性ごとの一般化レベルの上げ方の組み合わせの中で最も一般化・削除処理の少ないものを選択する問題になる. この問題は, 各属性ごとの一般化レベルのまとまりをノードとした場合, レベルの上げ方の組み合わせはラティス構造で表現でき, このラティス構造を効率よく探索する問題となる. 図 2 にラティス構造の例を示す.

この探索問題は, NP 困難であることが示されており [13], ヒューリスティクスに効率良く実行する方法が様々研究されている [3,7,8]. これらのアルゴリズムは, 「ラティス内のあるノードが k-匿名性を満たす場合, 任意の上位ノードも, k-匿名性を満たす」という性質に基づいてラティス内の探索空間を狭め, 高効率化を図っている.

2.3 データ管理方式

計算機を用いてデータを加工する場合, 2 次記憶装置である HDD や SSD から, 1 次記憶装置であるメインメモリにデータ転送し, 処理することが一般的である. 匿名加工対象となるデータは, 主に 2 次元の表で表現されるようなテーブル形式である. これらを 2 次元の表を計算機で管理する代表的な方法として, **行指向**および**列指向**でのデータ管理方法がある.

2.3.1 行指向データ管理

行指向データ管理とは, レコードを行単位でひとまとまりとして管理する方式のことをいう. 図 3 にその概念図を示す. 行指向データ管理の特徴として, レコードの追加や削除といったトランザクション処理が容易に行えることが利点である.

2.3.2 列指向データ管理

列指向データ管理とは, 列単位でひとまとまりとして管理する方式のことをいう. 図 3 にその概念図を示す. 列指向データ管理の特徴として, 少数列の集約処理といった, 分析用途に適していることが上げられる. また, 同一列でデー

© 2017 Information Processing Society of Japan

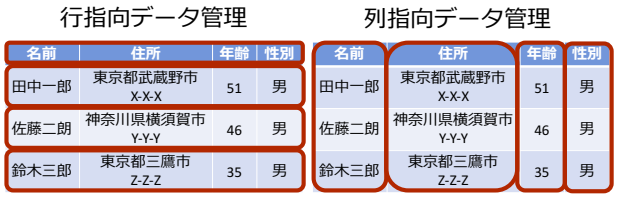

図 3 行指向および列指向データ管理の概念図. 赤枠で囲ったまとまり通り, 行指向は行単位, 列指向は列単位でデータをそれぞれ管理する.

タを管理することから, 似たような値でまとまるため, データ圧縮が効率的に行える利点がある.

2.4 コンピューティング方式

データを処理する際のアーキテクチャとして, 単一マシンによる処理と複数マシンによる処理 (分散コンピューティング) がある. 分散コンピューティングの代表的なものとして, MPI が上げられる. MPI は, 並列コンピューティングを行う上で標準化された規格であり, 分散コンピューティングにも頻繁に利用される. MPI は汎用的に使えるライブラリであるため, プログラミングの自由度が高い反面, ロック処理といった並列処理プログラミングの特有の難しさを考慮する必要がある.

他にも, プログラミングの難しさを軽減したものとして, Hadoop [1,5] や Spark [14] などがある. これらは, プログラミングの自由度を制限し, 特定の規則に則った形でプログラミングすれば, 安全に分散コンピューティングを行うことができ, また単一コンピューティングでボトルネックとなるディスク I/O 処理を, 複数マシンで分散し行うことで効率化を測っている.

これら分散処理方式は, 基本的に 1 台のコンピュータのメモリでは足りない際に, ディスクに内容を一時退避しなければならないケースに有効であることが多い.

3. 大規模なデータの匿名加工情報の作成に適したデータ処理方式の検討

大規模なデータを匿名加工する場合, それに適したデータ処理の方式を選択することで, 高効率にデータを加工することが可能となる. 本節では, 匿名加工情報に用いる**加工技法**と**データ規模**の 2 つの観点で, 適切なデータ管理およびコンピューティング方式を検討した.

3.1 加工技法ごとの適性

匿名加工情報の加工技法として, [16] や [15,20] で示されているようなデータ加工技法を組み合わせて適用する. これら加工技法を適用するにあたり, 節 2.3 で示した, 行指向および列指向どちらのデータ管理方法が適しているかを検討した. 適しているかどうかの観点は, その加工を行う際にレコード単位もしくは属性単位のどちらで連続して処理を行ったほうが効率的かを基準として判断した. 検討結果を表 2 に示す.

k-匿名化に関しては, 行/列指向という扱いにしている. k-匿名化アルゴリズムの多くは, 一般化処理と k-匿名性を満たすかどうかの判断を繰り返し行う. 一般化処理は, 列単位の処理が向いているが, k-匿名性の評価は, 行単位のほうが向いている. そのため, アルゴリズムの設計次第では, どちらが有利になるか変わってくる.

表 2 より, 行指向が 5 つ, 列指向が 7 つ適する技法があることから, 列指向データ管理が向いている技法が多いことがわかった.

3.2 データ規模に対する適性

匿名加工の対象となるデータは, 個人の特性に対するデータに加え, 位置情報の履歴といった横に長いデータであることが多い. ただし, [16] にあるように, 長い履歴データは一定の配慮をすることが望ましいと記載されているとおり, ある程度の期間に区切って匿名加工情報を作成するほうが良い. また, 匿名性の観点からも, データベース内に含まれる個人情報を全て含めたデータにせず, ある程度サンプリングされたデータを出すほうが良い.

これらを踏まえて, 想定されるデータ規模の検討を行った.

3.2.1 想定されるデータ規模

想定されるレコード数および履歴の長さから, データ規模を見積もる.

レコード数 日本の総人口は, [18] より 2017 年に 1.27 億人であるとわかる. もし 10 年前から現在までのデータを出したい場合でも, たかだか 1.37 億レコード程度を扱えれば良い [*2].
これらより, サンプリングを加味した場合 10 万〜1000 万程度, サンプリングを行わない場合は 1.37 億レコードを扱えれば良いといえる.

属性数 例えば 1 ヶ月分の移動履歴をまとめて出力したいと考えた場合, 緯度経度それぞれを 30 分おきに出力した場合 2880 属性, 1 時間おきに出力した場合 1440 属性必要となる. 属性数に関しては, データの種類により, もっと必要になるかもしれないが, 本稿では 5000 属性までで検討する.

これらレコード数および履歴の長さより, データ規模を見積もった結果が図 4 となる. これらの規模のデータをそのまま保持していることはなく, 往々にしてデータ圧縮されていることがほとんどである. 圧縮形式として Windows で利用される ZIP による圧縮を行った場合の想定データ規模を図 5 に示す.

[*2] 2007 年の総人口である 1.27 億人に対し, 人口動態調査より [17] およそ 1 年の出生数が 100 万人とした場合, 10 年でおよそ 0.1 億人出生する計算となり, 1.37 億人程度といえる

© 2017 Information Processing Society of Japan

表 2 匿名加工情報作成の際に用いる加工技法の一覧およびそれらの加工に適するデータ構造

加工技法	ソート	シャッフル	仮 ID 化	項目削除	レコード削除	一般化	トップ (ボトム) コーディング	サンプリング	ミクロアグリゲーション (グルーピング)	丸め	スワッピング	ノイズ付与	k- 匿名化
データ管理	行指向	行指向	列指向	列指向	行指向	列指向	列指向	行指向	行指向	列指向	列指向	列指向	行/列指向

属性＼レコード	10万	50万	100万	500万	1000万	5000万	1億	1.37億
10	0.01	0.05	0.10	0.51	1.02	5.12	10.24	14.04
25	0.03	0.13	0.26	1.28	2.56	12.81	25.61	35.09
50	0.05	0.26	0.51	2.56	5.12	25.61	51.22	70.18
75	0.08	0.38	0.77	3.84	7.68	38.42	76.83	105.3
100	0.10	0.51	1.02	5.12	10.24	51.22	102.4	140.4
250	0.26	1.28	2.56	12.81	25.61	128.1	256.1	350.9
500	0.51	2.56	5.12	25.61	51.22	256.1	512.2	701.8
750	0.77	3.84	7.68	38.42	76.83	384.2	768.3	1053
1000	1.02	5.12	10.24	51.22	102.4	512.2	1024	1404
2500	2.56	12.81	25.61	128.1	256.1	1281	2561	3509
5000	5.12	25.61	51.22	256.1	512.2	2561	5122	7018

図 4 レコード数と属性数を決めた際の実ファイスサイズを GB で表したもの. ファイル中のある属性の値は ascii で 10 文字で表されるとし, 属性間はカンマで区切られているとする. 例えば, 1000 万件 100 属性のデータは, 10.24GB とわかる.

属性＼レコード	10万	50万	100万	500万	1000万	5000万	1億	1.37億
10	0.003	0.02	0.03	0.16	0.33	1.64	3.28	4.49
25	0.01	0.04	0.08	0.41	0.82	4.10	8.20	11.23
50	0.02	0.08	0.16	0.82	1.64	8.20	16.39	22.46
75	0.02	0.12	0.25	1.23	2.46	12.29	24.59	33.68
100	0.03	0.16	0.33	1.64	3.28	16.39	32.78	44.91
250	0.08	0.41	0.82	4.10	8.20	40.98	81.96	112.3
500	0.16	0.82	1.64	8.20	16.39	81.96	163.9	224.6
750	0.25	1.23	2.46	12.29	24.59	122.9	245.9	336.8
1000	0.33	1.64	3.28	16.39	32.78	163.9	327.8	449.1
2500	0.82	4.10	8.20	40.98	81.96	409.8	819.6	1123
5000	1.64	8.20	16.39	81.96	163.9	819.6	1639	2246

図 5 レコード数と属性数を決めた際の ZIP 圧縮されたファイルのサイズを GB で表したもの. データは図 4 と同様. 圧縮率は事前の実験によりおおよそ 0.32 であった.

3.2.2 コンピューティング方式

昨今は, クラウドサービスなどを利用すれば手軽に計算機を用意することが可能である. 例えば, Amazon AWS EC2 で用意できる汎用サーバや高性能サーバとして,

汎用 CPU 32 コア, メモリ 160GB [*3]

高性能 CPU 64 コア, メモリ 976GB [*4]

などがある. 汎用サーバを用いた場合, 図 4,5 より, 未圧縮の場合 1000 万レコード 1000 属性程度 (1 億レコード 100 属性程度) であれば動くことがわかる. 圧縮したまま効率よくデータを扱える場合は, 1000 万レコード 2500 属性 (1 億レコード 250 属性) 程度まで可能となる. 高性能サーバを用いた場合は, 圧縮したまま効率良くデータを扱える場合, 1 億レコード 2500 属性程度まで動作可能であると期待できる.

これ以上のデータ規模を扱う必要がある場合は, 上記汎用サーバや高性能サーバだけでなく, 複数マシンによる加工処理を検討すべきである.

本稿では, 匿名加工情報作成時にサンプリングされている前提で, 単一マシンで扱える規模のデータ, おおよそ 100 億セル (1 億レコード 100 属性, 1000 万レコード 1000 属性) 相当を対象とする.

[*3] 2017 年 8 月 17 日現在の Amazon AWS EC2 m4.10xlarge に相当

[*4] 2017 年 8 月 17 日現在の Amazon AWS EC2 f1.16xlarge に相当

4. 大規模データに対する k-匿名化アルゴリズムの検討

前節を踏まえて, 表 1 に示す加工技法のアルゴリズムを実装すれば, 匿名化ライブラリが作成可能であると期待できる. しかしながら, 表 1 に示した加工技法のアルゴリズムのうち, k-匿名化は, 他の方式と比べてアルゴリズムが複雑であるため, アルゴリズムを工夫しなければ処理時間が多くなる恐れがある. 本節では, 大規模データに対する k-匿名化のアルゴリズムを検討する.

Prasser らは, データ規模が大きくなるにつれ, ラティス構造を探索する手段として, 幅優先探索ではなく, 深さ優先探索の方が効率よく処理可能であると報告している [11]. 本研究では, 深さ優先探索ベースのアルゴリズムをベースに, 並列化により効率良く実行できる方法を提案する [*5].

4.1 並列深さ優先探索 k-匿名化アルゴリズム

一般的に利用される再帰を用いた深さ優先探索アルゴリズムは, 並列化処理に向かない. 理由は, 再帰関数が直前の処理に依存して動作するものであり, 独立した処理とならないからである.

そこで, スタックを用いた再帰を伴わない深さ優先探索アルゴリズムをベースに, 並列実行な可能な深さ優先探索アルゴリズムを検討する. スタックを用いた再帰を伴わない深さ優先探索の k-匿名化アルゴリズムを, アルゴリズム 1 に示す.

アルゴリズム 1 では, 3 行以降 (WHILE ループ内) は独立な処理になっているゆえ, 複数スレッドによる並列実行が可能である. 更に並列化による効果を高めるには, 2 点に留意する必要がある.

1 つ目は, データの同期化処理による遅延の低減である. 例えば, 2, 4, 8 行目のスタックに積む, 取り出す処理や, 5, 10 行目の探索済みフラグをつける作業は, 複数スレッドが同じオブジェクトにアクセスすることが想定される. データの不整合が起きないよう, 最小限のロックを取得したり, スレッドセーフなオブジェクトを利用することが必要である. 例えば, Java 言語ではスレッドセーフなスタック構造として, BlockingDeque などが提供されており, これらを用いれば良い.

[*5] [11] では, Flash アルゴリズムが最も高速に動作すると報告しており, Flash アルゴリズムを並列化し高速にするのが素直である. しかし, Flash アルゴリズムは, 簡単に並列化できるわけではないと, 提案者自ら言及しており [7], 今回は並列化可能なアルゴリズムとして, 深さ優先探索ベースのアルゴリズムを選定した

© 2017 Information Processing Society of Japan

Algorithm 1 深さ優先探索による k-匿名化アルゴリズム

Input: ラティス構造
Output: 最適な k-匿名性を満たすノードの候補

```
1: 空のスタックを用意
2: ボトムノードをスタックに積む
3: while スタックが空でない場合 do
4:     スタックからノードを取り出す
5:     探索済みフラグをつける
6:     ノードに示す一般化レベルで一般化処理
7:     if k-匿名性を満たしていない then
8:         現在のノードに接続している親ノードのうち, 未探索な親
           ノードをスタックに積む
9:     else
10:        現在のノードを最適な k-匿名性を満たすノードの候補に加
           える
11:        現在のノードに接続している親ノードはこれ以上探索の必
           要がないため, 再帰的に探索済みフラグを付ける.
12:    end if
13: end while
```

2 つ目は, 一般化処理の効率化である. 節 2.3.2 で示した列指向データ管理を利用しかつデータを**圧縮**している場合, 6 行目の一般化処理は, データを解凍した後に実行する必要がある. 解凍処理は, 時間を要するため, 出来る限り実行回数を減らすことが望ましい.

4.1.1 一般化処理の事前計算および符号サイズの最適化による高効率化

解凍処理の実行回数を減らすアプローチとして, 事前にすべての一般化レベルのデータを作成する方法が考えられる. 一般化処理の際 (アルゴリズム 1 の 6 行目) は, 作成したデータを参照するだけで良い. 図 2 に示したラティス構造の場合を考える. この場合, 日付は 3 つ, 性別は 2 つ, 住所は 5 つの一般化レベルのデータ, 計 10 つのデータを作ることとなる. 例えば日付属性の 3 つの一般化レベルのデータを作成する場合, 日付列のデータを一度解凍し, 2 つの一般化レベルのデータを作成 (レベル 0 は生データなのでそのまま) する. 同様に性別や住所も 1 度だけそれぞれ解凍処理を行い, 一般化処理を行う. ラティス構造の各ノードを辿る場合は, 作成した各一般化レベルのデータを参照する形で済む.

先程述べた方式では, 予め一般化したデータを保持することから, メモリ使用量が増えてしまう. 例えばレコード数が 1 億件, 図 1 に示した一般化階層, 1 文字を 2byte で表現し各値が平均 5 文字で表現されている場合, メモリ使用量は, 1 億 (件) × 10(byte) × 10(個の一般化されたデータ (日付 3, 性別 2, 住所 5 の計 10)) = 18GB のメモリを消費することになってしまう.

そこで, 文字列で表現せずコード化して保持することで, 効率化を測る. 具体的には, 一般化した後のユニークな値の数が 255 以下なら 1byte, 65,535 以下なら 2byte, 4,294,967,295 以下なら 4byte で, それぞれ値を保持することで, 定数倍であるが, メモリの消費量を抑えることができる. 例えば, レコード数 1 億件な住所データで, 図 1 に示す住所の一般化階層の場合, レベル 0 は 4byte, レベル 1 は 4byte, レベル 2 は 4byte, レベル 3 は 1byte, レベル 4 は 1byte で表現可能であり, およそ 1.3GB 程度で済む. 対して, 文字列で平均 10 文字で表現した場合は 9.3GB 程度と, およそ 7 倍効率よくデータを保持できてることがわかる.

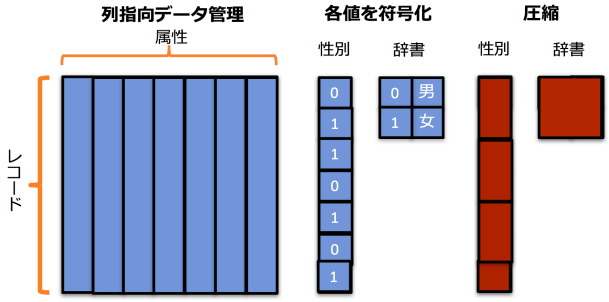

図 6 列指向データ管理の実装イメージ図

1 億件規模ともなると, GB 単位のデータ量となることから, 定数倍の改善でも顕著に性能に響くことがわかる.

5. ライブラリの実装

前節までの検討を受け, **単一マシン**で動作する匿名化ライブラリを実装した. データ構造として**列指向データ管理**を用い, k-匿名化のアルゴリズムとして並列深さ優先探索アルゴリズムを実装した. k-匿名化アルゴリズムに加えて, 表 1 の技法や, ランダム化による k-匿名性を達成することのできる Pk-匿名化 [6] も実装した. 他に, データの外れ値を検出する機能も実装した. 本稿では紙面の都合上, 加工技法については, k-匿名化に絞り議論する. 実装した他の技法の性能や詳細については別の機会で示す.

列指向データ管理の際は, 図 6 に示すように,

- 重複する値を辞書で管理する,
- 列や辞書を分割して圧縮する,

などの工夫を行っている. 列単位の圧縮や解凍の際は, 予めデータが分割されているため, 分割毎に並列に圧縮解凍処理を行い, 効率を上げる実装を行っている. 圧縮にはデファクトスタンダードである zlib を用い, 並列アルゴリズムには Java1.7 より採用されている Work Stealing アルゴリズムを用いた.

6. 実験

6.1 比較対象

比較として, ARX Powerful Anonymization Tool [4](以下, ARX) を対象とする. ARX はオープンソースで公開されている匿名化ツールである. k-匿名化のアルゴリズムとして, Flash というアルゴリズムを搭載している. Flash は, [11] より他のグローバルリコーディングによる k-匿名化アルゴリズムと比べて最も高速に動作すると報告されている. データは行指向で管理しており, 重複する値を辞書

© 2017 Information Processing Society of Japan

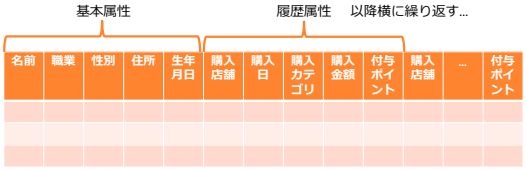

図 7 購買履歴を模した人工データ

表 3 図 7 に示すデータの各値の詳細

属性値	取りうる値
名前	5000 種類の名字と 5000 種類の名前とのランダムな掛け合わせ
職業	1,...,24 のいずれかで符号化済み.
性別	男 or 女
住所	5000 種類の住所 (丁目や番地などを除く) とランダムな丁目, 番地のランダムな掛け合わせ
生年月日	1950 年 1 月 1 日以降で, 年-月-日のような記述
購入店舗	A,...,Z の 1 文字
購入日	年-月-日 時-分 で記述 (ある一ヶ月間)
購入カテゴリ	1,...,24 のいずれか
購入金額	1000,...,100000 のいずれか
付与ポイント	0,...,10000 のいずれか

表 4 作成した人工データの一覧

レコード数	属性数	ファイルサイズ	属性ごとの値の平均重複率
100,000	100	67.7MB	0.828
1,000,000	100	677.7MB	0.954
10,000,000	100	6.5GB	0.983
100,000,000	100	64GB	0.989

で管理することにより, メモリ効率を良くする工夫が行われている. 本ライブラリと同様に Java 言語で実装されており, 詳細な実装方法は, [4] を参照されたい. なお今回は, ARX の GUI は利用せず, ライブラリのみを用いて実装し評価を行った.

6.2 実行環境

一般的な計算機サーバを用意し, 検証を行った. 検証に用いたサーバのスペックを以下に示す.

- CPU : Intel Xeon E5-2609v2 (Ivy Bridge 2.5GHz 4 コア) x2 (計 8 コア)
- Memory : DDR3 384GB
- HDD : 500GB (RAID1)
- OS : Red Hat Linux 7.2
- JVM : OpenJDK 1.8 JVM

JVM に割り当てるヒープ領域は実メモリの 1/4 である 96GB として実験を行った.

6.3 対象データ

実装したライブラリの性能を測定するため, データの規模を任意で変更可能な人工データにより評価を行う. 人工データとして, 購買履歴を模したデータを作成した. データの属性を図 7 および各値をの取りうる値を表 3 に示す. また, 測定対象とするデータについて表 4 に示す.

今回は, QI を職業, 性別, 住所, 生年月日とする. それぞれ一般化階層について, 性別, 生年月日は, 図 1 と同様で, 住所は図 1 のレベル 4 を除いた高さ 4 の一般化階層を用意した. 職業に関しては, レベル 1 が [1-6],[7-12],[13-18],[19-24], レベル 2 が [1-12], [13-24] となる階層構造を用意した.

表 5 読み込み時間の比較

レコード	属性数	ARX	提案
100,000	100	3.4[s]	2.2[s]
1,000,000	100	30[s]	21[s]
10,000,000	100	332[s]	220[s]
100,000,000	100	NA	2213[s]

表 6 実メモリの比較

レコード数	属性数	ARX	提案
100,000	100	136.4MB	52.4MB
1,000,000	100	658.1MB	224.6MB
10,000,000	100	NA	NA
100,000,000	100	NA	NA

6.4 結果・考察

6.4.1 読み込みの実行時間

本ライブラリや ARX は一般的にデータを扱う形式である CSV に対応している. 本節では, CSV 読み込みをし, 匿名化アルゴリズムを実行前までの処理時間および, メモリ使用量の比較を行う.

読み込み時間 CSV 読み込みにかかる時間を計測した結果を表 5 に示す. ARX と比べて遜色なく (ARX は, uniVocity[*6] という CSV パーサーを用いている. Java の中で最も速い CSV パーサーの 1 つである), むしろ高速に読み込みができている. 理由として, zlib による圧縮を行っているが, 列指向の特性を活かし, 列ごとに並列化してデータ格納を行っており, 効率を上げているからである.

メモリ使用量 読み込み後のオブジェクトのメモリ使用量を表 6 に示す. なお, オブジェクトのメモリ使用量を測るために, java.sizeof[*7] を用いた. ARX で用いている圧縮では, 10 万レコードの場合, 実ファイルの 2 倍程度のメモリ使用量で, 100 万レコードの場合実ファイルサイズ程度のメモリ使用量であった. 比べて提案手法は, zlib による圧縮の効果により, 10 万レコードの場合は, 実ファイルサイズの 70%割り, 100 万レコードの場合は実ファイルサイズの 33%となっており, はるかにコンパクトに圧縮されていることがわかる.

一部 NA となっている部分は, java.sizeof でエラーが発生し, 計測できなかった. jvm のメモリ状況を確認したところ, それぞれリニアにメモリ使用量が増えていることを別途確認した.

6.4.2 *k*-匿名化アルゴリズムの実行時間

本節では, 匿名化アルゴリズムに生じる処理時間の比較

*6 https://github.com/uniVocity/univocity-parsers
*7 http://sizeof.sourceforge.net/

© 2017 Information Processing Society of Japan

表 7 k-匿名化アルゴリズム (QI に対して. すなわち 4 属性) の実行時間の比較

レコード	属性数	ARX	提案
1,000,000	100	15[s]	32[s]
10,000,000	100	437[s]	274[s]
100,000,000	100	NA	6118[s]

を行う. 3-匿名性で, 10%の削除を許容率で, k-匿名化アルゴリズムを実行した. 実行時間を表 7 に示す. 100 万レコードの場合, ARX のほうが提案より 2 倍高速であることがわかる. これは, 提案方式の zlib 圧縮の解凍にかかる時間や, 並列処理の同期の時間がボトルネックになっているといえる.

1000 万レコードの場合は, 提案のほうが 1.5 倍高速であるとわかった. この規模になると, 並列化の同期処理に比べて, 一般化の処理のほうが時間を要したことから, 並列化による効果が顕著に現れたといえる. ARX は 1 億レコードでは動作できず, 本ライブラリでは動作し, およそ 1 時間 42 分で終了した.

7. おわりに

本研究では, 匿名加工情報を作成するにあたり, 必要な機能に適したアーキテクチャの検討を行った. また, データ規模が大きい場合を想定し並列化による処理の高速化も検討した. そして, 本検討結果を実装し, 評価を行った. 結果, 従来最も速い匿名化アルゴリズムを備えたライブラリと比べおよそ半分以下のメモリ使用量で, 1000 万件規模のデータの場合 1.5 倍高速に動作することを確認した. 加えて 1 億件では既存のライブラリでは実行できなかったが, 本ライブラリでは実行でき, k-匿名化処理が 1 時間 42 分で完了するという実用的な時間での動作を確認した.

今後の課題として, 分散処理による方式との実行時間の比較検討があげられる. 加えて, 本稿では紙面の都合上記載できなかった k-匿名化以外の加工技法との組み合わせをした場合の実行時間の評価が上げられる.

参考文献

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pp. 10–10. USENIX Association, 2004.

[2] Cynthia Dwork. Differential privacy. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming*, ICALP'06, pp. 1–12, 2006.

[3] Khaled El Emam, Fida Kamal Dankar, Romeo Issa, Elizabeth Jonker, Daniel Amyot, Elise Cogo, Jean-Pierre Corriveau, Mark Walker, Sadrul Chowdhury, Regis Vaillancourt, et al. A globally optimal k-anonymity method for the de-identification of health data. *Journal of the American Medical Informatics Association*, Vol. 16, No. 5, pp. 670–682, 2009.

[4] Prasser Fabian, Kohlmayer Florian, Lautenschlaeger Ronald, and A. Kuhn Klaus. Arx a comprehensive tool for anonymizing biomedical data. In *AMIA Annual Symposium Proceedings*, pp. 984–993, 2014.

[5] Apache Foundation. Apache hadoop. http://hadoop.apache.org/.

[6] Dai Ikarashi, Ryo Kikuchi, Koji Chida, and Katsumi Takahashi. k - anonymous microdata release via post randomization method. In *International Workshop on Security 2015*, pp. 225–241, 2015.

[7] Florian Kohlmayer, Fabian Prasser, Claudia Eckert, Alfons Kemper, and Klaus A Kuhn. Flash: efficient, stable and optimal k-anonymity. In *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*, pp. 708–717. IEEE, 2012.

[8] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pp. 49–60, 2005.

[9] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. Mondrian multidimensional k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering*, pp. 25–25, 2006.

[10] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pp. 24–24. IEEE, 2006.

[11] Fabian Prasser, Florian Kohlmayer, and Klaus A Kuhn. A benchmark of globally-optimal anonymization methods for biomedical data. In *Computer-Based Medical Systems (CBMS), 2014 IEEE 27th International Symposium on*, pp. 66–71. IEEE, 2014.

[12] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, Vol. 10, No. 05, pp. 557–570, 2002.

[13] Ryan Williams, et al. On the complexity of optimal k-anonymity. In *Proc. 23rd ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS*, 2004.

[14] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pp. 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[15] 経済産業省. 事業者が匿名加工情報の具体的な作成方法を検討するにあたっての参考資料(「匿名加工情報作成マニュアル」) ver1.0. http://www.meti.go.jp/policy/tokumeikakou.pdf.

[16] 個人情報保護委員会. 匿名加工情報「パーソナルデータの利活用促進と消費者の信頼性確保の両立に向けて. https://www.ppc.go.jp/files/pdf/report_office.pdf.

[17] 厚生労働省. 人口動態調査. http://www.mhlw.go.jp/toukei/list/81-1.html.

[18] 総務省統計局. 人口推計-平成 29 年 7 月報-. http://www.stat.go.jp/data/jinsui/pdf/201707.pdf.

[19] 大角良太, 高橋克巳. QA で理解する!パーソナルデータの匿名加工と利活用. 清文社, 2017.

[20] 匿名加工情報に関する技術検討ワーキンググループ. 匿名加工情報の適正な加工の方法に関する報告書 2017 年 2 月 21 日版. http://www.nii.ac.jp/research/reports/pd/report-kihon-20170221.pdf.

© 2017 Information Processing Society of Japan