一般発表 | Track 2: ビッグデータ基盤技術・セキュリティ・プライバシ

🏛2024年3月1日(金) 10:00 ~ 12:10 | 🏛T2-B オンライン（Zoom Events)

時系列データ処理

座長:渡辺 陽介(名古屋大学)
コメンテータ:喜田 拓也(北海学園大学)

10:00 ~ 10:05
イントロダクション

10:05 ~ 10:20
[T2-B-7-01] クエリ長に依存しない多次元時系列データに対する類似問合せ手法の提案

*安田 裕真[1]、塩川 浩昭[2] (1. 筑波大学　大学院理工情報生命学術院、2. 筑波大学　計算科学研究センター)

10:20 ~ 10:35
[T2-B-7-02] NexmonによるCSIベースのスマートデバイス状態推定

*原田 海斗[1]、寺本 京祐[2]、野村 祐一郎[3]、峰野 博史[3,4] (1. 静岡大学 情報学部 情報科学科、2. 静岡大学 大学院 総合科学技術研究科 情報学専攻、3. 静岡大学 学術院 情報学領域、4. 静岡大学 グリーン科学技術研究所 フィールドインフォマティクス研究コア)

10:35 ~ 11:00
[T2-B-7-03] LSiX:ストリームデータに関する複数連続的集約

*川上 隼[1]、Bou Savong[2,1]、天笠 俊之[2,1] (1. 筑波大学、2. 計算科学研究センター)

11:00 ~ 11:25
[T2-B-7-04] 大規模データストリームに対する高速なS-FINCHクラスタリング

*牛尼 索造[1]、藤原 靖宏[2]、塩川 浩昭[3] (1. 筑波大学 情報学群情報科学類、2. NTTコミュニケーション科学基礎研究所、3. 筑波大学 計算科学研究センター)

11:25 ~ 11:50
[T2-B-7-05] 【技術報告】東芝の時系列データ処理基盤技術(GridDB)ならびにデータ仮想化エンジン(PGSpider)

浪岡保男、服部雅一、相川 恵 (株式会社東芝)

クエリ長に依存しない多次元時系列データに対する類似問合せ手法の提案

安田　裕真†　　塩川　浩昭††

† 筑波大学大学院理工情報生命学術院　〒 305–8577 茨城県つくば市天王台 1-1-1
†† 筑波大学計算科学研究センター　〒 305–8577 茨城県つくば市天王台 1-1-1
E-mail: †yasuda@kde.cs.tsukuba.ac.jp, ††shiokawa@cs.tsukuba.ac.jp

あらまし　多次元時系列データにおける類似部分シーケンス問合せとは，クエリとして入力された多次元時系列データに対して類似度の高い部分シーケンスをデータベースから探索する問題であり，医学やスポーツ科学の分野などで広く利用されている．問合せの効率化として，部分シーケンス長で多次元時系列データを分割することでクラスタリングを行う手法がある．この手法には，部分シーケンス長とクエリ長が異なると問合せを行えないという問題点がある．そこで本論文では，クエリ長に依存しない類似部分シーケンス問い合わせ手法を提案する．実データを用いた性能評価を行い，従来手法と比較して提案手法は高速かつ正確にクエリ長に依存しない類似部分シーケンス問合せを行えることを示す．

キーワード　時系列データ処理，データ構造・索引，問合せ処理

1　序　　論

多次元時系列データとはある現象の時間的な変化を連続的に観測して得られた多次元の実数値からなる系列（シーケンス）であり，医学やスポーツ科学の分野において，重要な要素技術となっている [1]．近年，多次元時系列データ解析において，クエリと類似した部分シーケンスを検索する類似部分シーケンス問合せ処理技術が注目を集めている．類似部分シーケンス問合せとは，クエリと多次元時系列データの取りうる全ての部分シーケンスとの間で類似度を計算し，類似度が高くなった部分シーケンスを類似部分シーケンスとして出力する処理である．類似部分シーケンス問合せでは多次元時系列データ内に存在する全ての部分シーケンスを検索の対象とするため，長いシーケンスに対して膨大な問合せ処理時間を要する問題がある．

これまで，類似部分シーケンス問合せ手法の様々な研究 [2] [3] がなされているが，いずれも 1 次元の時系列データに対する手法であり，多次元への応用はなされていなかった．また，多次元時系列データを対象とした研究 [4] [5] もなされてきたが，これらは多次元時系列データから最も類似した部分シーケンスのペアを検出するという手法であり，クエリに類似する部分シーケンスを検索することはできなかった．

これらの問題を解決するための手法として，局所性鋭敏型ハッシュ関数（LSH）を用いた LSH 手法が提案された [6]．LSH 手法は，多次元時系列データとクエリが与えられたときに，クエリとのピアソン相関が閾値 θ 以上の部分シーケンスを高速かつ正確に出力する．具体的には，天方らによる先行研究 [7] を多次元時系列データへと拡張し，多次元時系列データとクエリを事前に次元削減することで，全ての部分シーケンスを探索することなく検索を行う．

しかし，LSH 手法にはクエリ長と部分シーケンス長が一致しないと検索を行えないという問題がある．LSH 手法では，多次元時系列データの各部分シーケンスに対して LSH を用いてハッシュ値を計算し，ハッシュ値を基にクラスタリングを行っているため，クエリ長と部分シーケンス長が異なるとグループ検索を行うことができなかった．

そこで，本研究では LSH 手法を拡張することで，クエリ長に依存しない類似部分シーケンス問合せ手法の提案を行う．我々は，クエリ長と部分シーケンス長が異なる 3 つのケースに対して対応策を提案する．本論文では実データを用いた提案手法の評価実験を行い，提案手法が LSH 手法と比較して同程度の速さと精度でクエリ長に依存しない類似部分シーケンス問合せを行えることを確認した．

本論文の構成は以下の通りである．2 節で前提となる知識と本研究で取り扱う問題について説明し，3 節で LSH 手法を紹介する．4 節で提案手法について述べ，5 節で評価実験結果を示す．6 節で本研究のまとめを行う．

2　事 前 準 備

多次元時系列データ $\boldsymbol{T}$ とは z 正規化された時系列データの集合であり $\boldsymbol{T} = [T^{(1)}, T^{(2)},...,T^{(d)}]$ と表記する．d は多次元時系列データ $\boldsymbol{T}$ の次元数，$T^{(i)}$ は長さ n の 1 次元時系列データであり，$\boldsymbol{T} \in \mathcal{R}^{(d \times n)}$ である．ここで，$T^{(j)}$ において，先頭から i 番目の長さ m である部分シーケンスを $T^{(j)}_{i,m} = [t^{(j)}_i, t^{(j)}_{i+1}, \ldots, t^{(j)}_{i+m-1}]$ と表す．ただし，$t^{(j)}_{i+k}$ は $T^{(j)}_{i,m}$ の $i+k$ 番目の実数値を表す．また，$\boldsymbol{T_{i,m}}$ は $\boldsymbol{T}$ 内に含まれる先頭から i 番目の長さ m である部分シーケンス集合であり $\boldsymbol{T_{i,m}} = [T^{(1)}_{i,m}, T^{(2)}_{i,m}, \ldots, T^{(d)}_{i,m}]$ である．また，$\boldsymbol{T_{i,m}}$ を $(d \times m)$ 行列とみなしたとき，k 列目の列ベクトルを $\mathbf{t}^k_{i,m}$ とすると $\mathbf{t}^k_{i,m}=(t^{(1)}_{i+k-1}, t^{(2)}_{i+k-1}, \ldots, t^{(d)}_{i+k-1})^T$ と表すことができる．クエリデータ q とは，次元数 d，長さ m の実数値から構成される多次元時系列データであり，$q \in \mathcal{R}^{(d \times m)}$

である．

多次元時系列データに対する類似部分シーケンス問合せとは $\boldsymbol{T}$ とクエリデータ q が与えられたとき，クエリと類似した部分シーケンスを $\boldsymbol{T}$ から検出することである．本研究では部分シーケンス間の類似度としてピアソン相関を採用する．定義は以下の通りである．

定義1（ピアソン相関）．2つの部分シーケンス $\boldsymbol{T}_{a,m}$ と $\boldsymbol{T}_{b,m}$ の間のピアソン相関は以下のように求める．

$$\rho(\boldsymbol{T}_{a,m}, \boldsymbol{T}_{b,m}) = 1 - \frac{|dist(\boldsymbol{T}_{a,m}, \boldsymbol{T}_{b,m})|^2}{2m}$$

ただし，$dist(\mathbf{T}_{a,m}, \mathbf{T}_{b,m})$ は $\mathbf{T}_{a,m}$・$\mathbf{T}_{b,m}$ 間のユークリッド距離であり，以下のように計算する．

$$dist(\boldsymbol{T}_{a,m}, \boldsymbol{T}_{b,m}) = \sqrt{\sum_{k=1}^{m}(t_{a,m}^k - t_{b,m}^k)^2}$$

本研究で対象とする類似部分シーケンス問合せ問題を以下のように定義する．

定義2（類似部分シーケンス問合せ問題）．多次元時系列データ $\boldsymbol{T}$，クエリデータ q，閾値 $\theta \in [0, 1]$ が与えられたとき，ピアソン相関 $\rho(\boldsymbol{T}_{i,m}, q) \geqq \theta$ を満たすような全ての部分シーケンス $\boldsymbol{T}_{i,m}$ を $\boldsymbol{T}$ から検索する．

3 先行研究 LSH 手法

本節では LSH 手法 [6] について説明する．LSH 手法の目的は，多次元時系列データに対して高速かつ正確に類似部分シーケンス問合せを行うことである．3.1 節から 3.3 節で，LSH 手法の詳細な説明を行う．

3.1 ハッシュ処理

ハッシュ処理では多次元時系列データ $\boldsymbol{T}$ 内の各部分シーケンス $\boldsymbol{T}_{i,m}$ に LSH を適用し，ハッシュ値を得る．LSH 手法では，先行研究 [7] で提案された LSH を拡張し，以下のハッシュ関数を用いる．

定義 3（LSH）．部分シーケンス $\boldsymbol{T}_{i,m} \in \mathcal{R}^{(d\times m)}$ に対して，LSH $h(\boldsymbol{T}_{i,m})$ を以下のように定義する．

$$h(\boldsymbol{T}_{i,m}) = \frac{(\boldsymbol{T}_{i,m} \cdot \mathbf{a})^T \cdot \mathbf{b} + cw}{w}$$

ただし，$\mathbf{a} \in \mathcal{R}^{(d\times 1)}$，$\mathbf{b} \in \mathcal{R}^{(m\times 1)}$ はそれぞれ各要素が正規分布に従うランダムな値を持つ列ベクトルである．c は $[0, w)$ から選ばれたランダムな実数であり，w は定数である．上式の通り，LSH $h(\boldsymbol{T}_{i,m})$ は最終的に，1つの実数値を出力する．

ここで，ピアソン相関の閾値を θ とおくと，本研究では $\rho(\boldsymbol{T}_{a,m}, \boldsymbol{T}_{b,m}) \geqq \theta$ となるような類似部分シーケンスに注目し，これらのハッシュ値が類似するようにしたい．そこで先行研究 [7] に従い，$w = \sqrt{2m(1-\theta)}$ とする．

LSH 手法は定義 3 に示したハッシュ関数を L 個用意して，各部分シーケンスに対して L 個のハッシュ値 H=[$h_1(\boldsymbol{T}_{i,m})$, $h_2(\boldsymbol{T}_{i,m})$, ..., $h_L(\boldsymbol{T}_{i,m})$] $(1 \leqq i \leqq n-m+1)$ を生成する．これにより，サイズ $L\times(n-m+1)$ の行列であるハッシュ値系列 H を得る．

3.2 グループ化

グループ化では，ハッシュ値系列 H=[$h_1(\boldsymbol{T}_{i,m})$, $h_2(\boldsymbol{T}_{i,m})$, ..., $h_L(\boldsymbol{T}_{i,m})$] $(1 \leqq i \leqq n-m+1)$ の各部分シーケンス同士のピアソン相関を求め，相関が強い部分シーケンスを同じグループに格納する．具体的には H の各列間のピアソン相関を求める．このとき，2つの部分シーケンスが $\rho \geqq \theta$ となる場合，同じグループに格納していく．この操作を全ての部分シーケンスの組合せに対して実行し，ピアソン相関の大きな部分シーケンス同士を同じグループに格納する．グループ化によって作成されたグループの集合を *H-group* と呼ぶ．

3.3 検　　索

クエリデータ q が到着したら，時系列データと同様のハッシュ処理を q に行い，H_q=[$h_1(q)$, $h_2(q)$, ..., $h_L(q)$] とする．H_q と *H-group* の各グループとのピアソン相関を求め，閾値 θ よりも大きいグループを検索．検索されたグループの集合を H_q*-group* とする．ここで H_q*-group* 内の各部分シーケンスと H_q を元のデータに復元する．同様に復元したクエリと元の部分シーケンスのピアソン相関を計算し，ピアソン相関が閾値 θ よりも大きい部分シーケンスを出力する．

4 提 案 手 法

本節では，LSH 手法を拡張した提案手法の説明を行う．提案手法の目的は，LSH 手法においてクエリ長が部分シーケンス長と一致しない場合でも，類似部分シーケンス問合せを行えるようにすることである．クエリ長と部分シーケンス長が異なるケースとして，以下の 3 つのケースが考えられる．

ケース1：クエリ長が部分シーケンス長より長く，部分シーケンス長を組み合わせることでクエリ長と一致するケース

ケース2：クエリ長が部分シーケンス長より長く，部分シーケンス長を組み合わせてもクエリ長と一致しないケース

ケース3：クエリ長が部分シーケンス長より短いケース

これら 3 つのケースそれぞれに対して対応策を提案する．ここで，クエリ長を *Query-m*，部分シーケンス長を *Sub-m* と表記することとする．

4.1 ケ ー ス 1

クエリ長 *Query-m* が部分シーケンス長 *Sub-m* より長く，*Sub-m* を組み合わせることで *Query-m* と一致するケースでは，クエリを *Sub-m* に合わせて分割を行い，分割したクエリごとに検索を行う．具体例として図 1 に示す．図のインデックスは，部分シーケンスが始まる位置を表している．まず，(1) のように到着したクエリを *Sub-m* で分割して，分割したクエリごとにグループの検索を行う．次に，(2) のように分割したクエリごとに検索されたグループ内の部分シーケンスのインデックス間の差を計算する．インデックス間の差が *Sub-m* と一致する場合，その 2 つの部分シーケンスは連続した位置関係にあるこ

とがわかる．この性質を利用して部分シーケンスを合成することで，*Query-m* と等しい *Sub-m* の類似部分シーケンスを検索することができる．クエリに類似した部分シーケンス検索後の操作は，LSH 手法と同じである．

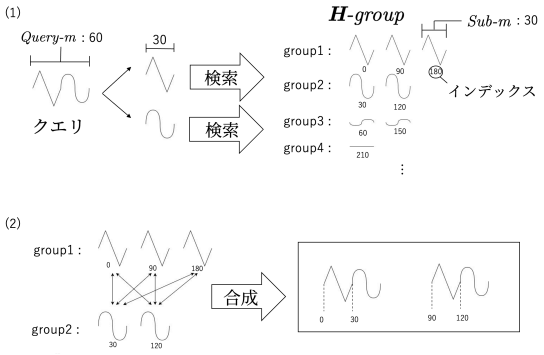


図 1　ケース 1 の例

4.2 ケース 2

クエリ長 *Query-m* が部分シーケンス長 *Sub-m* より長く，*Sub-m* を組み合わせても *Query-m* と一致しないケースでは，クエリを *Sub-m* の部分と余剰部分に分割を行い，余剰部分の長さに応じて操作を分ける．具体的には，余剰部分の長さが *Sub-m* の $\frac{1}{2}$ 以上のときと $\frac{1}{2}$ 未満のときで操作を分ける．

余剰部分が *Sub-m* の $\frac{1}{2}$ 以上のときは，余剰部分が *Sub-m* と等しくなるように余剰部分の平均値で埋める．具体例として図 2 の (1) に示す．図のように，余剰部分の長さが 27 と，*Sub-m* の $\frac{1}{2}$ 以上のときは，余剰部分の平均値で埋める．この理由は，余剰部分の特徴を失わせないためである．LSH を用いたハッシュ処理を行うため，余剰部分に全く関係のない乱数で埋めてしまうと，本来の余剰部分と類似していたグループを検索することができない．そこで，余剰部分の平均値で埋めることで，特徴を最低限残すようにする．余剰部分を埋めた後の操作は，ケース 1 と同じである．

余剰部分が *Sub-m* の $\frac{1}{2}$ 未満のときは，余剰部分を除外する．具体例として図 2 の (2) に示す．図のように，余剰部分の長さが 10 と，*Sub-m* の $\frac{1}{2}$ 未満のときは，余剰部分を除外する．この理由は，グループ検索の精度を下げないためである．*Sub-m* の $\frac{1}{2}$ 未満の余剰部分の平均で埋めてしまうと，余剰部分の特徴を捉えるには不十分な特徴のみが残ってしまい，本来検索されるであろうグループと異なるグループが検索されてしまう．このことを避けるためにも，余剰部分を除外して検索を行うことで，検索時間は長くなってしまうが精度の面では保証することができる．余剰部分を除外した後の操作は，ケース 1 と同じである．

4.3 ケース 3

クエリ長 *Query-m* が部分シーケンス長 *Sub-m* より短いケースでは，いかなる場合でもクエリの平均でクエリを埋める必要がある．与えられたクエリ以外にグループ検索に使える要素は無いため，精度が下がってしまってもクエリの平均で埋めて検索を行う必要がある．

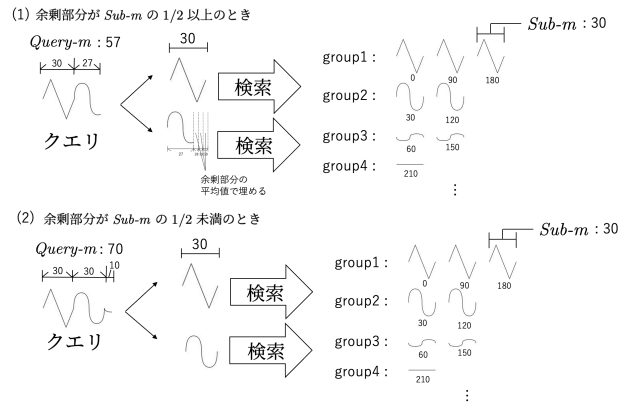


図 2　ケース 2 の例

5 評価実験

提案手法の検索時間と処理精度の評価を行うために，比較手法として 3 つの手法を用意する．1 つ目の手法は，クエリデータと多次元時系列データの全ての部分シーケンスとのピアソン相関を計算するという手法である．これをベースライン手法と呼ぶ．2 つ目の手法は，離散フーリエ変換（DFT）を用いて次元削減を行う手法 [3] を多次元に応用した手法を用いる．文献 [3] では，1 次元の時系列データに対してクエリとの類似度が高い類似部分シーケンス検索を行っているため，この手法を多次元に応用する．具体的には，多次元時系列データの各次元に対して文献 [3] の手法を用いて類似部分シーケンス検索を行い，各次元から得られる結果を組み合わせることで類似部分シーケンス検索を行う．この手法を DFT 手法と呼ぶ．3 つ目の手法は，3 章で紹介を行った LSH 手法 [6] である．検索時間と処理精度を評価するために，実データを利用する．データセットの詳細は 5.1 節に述べる．5.2 節では提案手法と比較手法の問い合わせ処理の評価実験の結果を示す．5.3 節では提案手法のクエリ長と部分シーケンス長の関係による，問合せ処理時間と処理精度の変化について検証を行う．

5.1 実験データセット詳細

実データセットは論文やデータセットを公開しているサイトから引用したデータである．実データセットの詳細として，以下の表 3 に示す．

表 1　実データセットの詳細

データセット	n	d	詳細
Matrix profile Dataset	550	3	Matrix Profile の論文内で用いられたデータセット [9]
Basket Dataset	10,000	6	バスケットボールの運動データセット [10]
Smartphone Dataset	152888	9	スマートフォンの位置情報データセット [11]
Mt.Tsukuba Dataset	158422	11	筑波山の気象データセット [12]

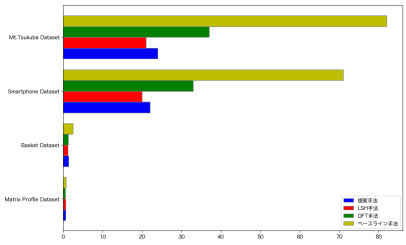


図 3　ケース 1

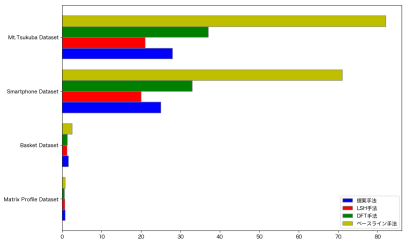


図 4　ケース 2

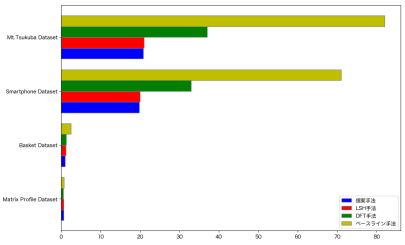


図 5　ケース 3

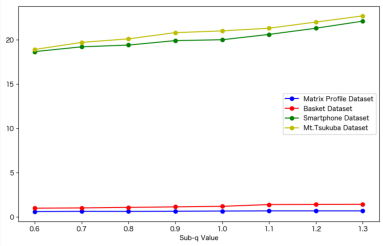


図 6　問合せ処理時間の変化

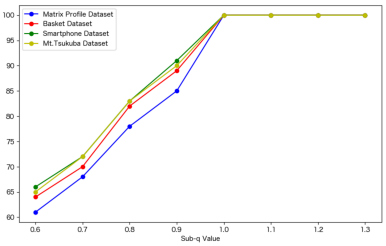


図 7　処理精度の変化

5.2　評価実験結果

本節では，提案手法と比較手法を用いて問合せ処理の評価実験を行う．評価方法として問合せ処理の実行時間と処理精度の計測を行う．

5.2.1　問合せ処理の実行時間

提案手法と比較手法の問合せ処理時間の検証を行う．実験結果を図 3,4,5 に示す．結果から，ケース 1,2 のような LSH 手法よりも計算量が増加するケースにおいては，LSH 手法よりも多くの問合せ処理時間を要するが，計算量が増加しないケース 3 においては，LSH 手法と同等の速さで問合せ処理を行えることがわかる．また，ケース 1,2,3 のいずれにおいても，ベースライン手法と DFT 手法よりも高速に問合せ処理を行えることがわかる．これは，提案手法の計算量が，DFT 手法とベースライン手法の計算量よりも少ないことを表している．

5.2.2　問い合せ処理の処理精度

提案手法と比較手法の問合せ処理の処理精度の検証を行う．検証準備として実データの 1 番目の部分シーケンスをクエリデータ q として抜き出す．1 番目の部分シーケンスを抜き出した実データに対して，ベースライン手法を用いてクエリとのピアソン相関を求め，ピアソン相関係数が閾値 θ 以上の部分シーケンスをあらかじめ検索しておく．この実データにおいて提案手法と比較手法を用いて類似データの検索を行い，処理精度の検証を行う．提案手法と LSH 手法では $L = 10$ でハッシュ処理を行う．また，提案手法ではクエリ長が部分シーケンス長の 2 倍であると仮定して検索を行う．本研究では処理精度として，適合率と再現率を用いる．ベースライン手法によって検索された正解データ集合を G，提案手法によって検索された部分シーケンス集合を R とすると，適合率と再現率はそれぞれ以下のように表せる．

$$\text{適合率} = \frac{|G \cap R|}{|R|}, \quad \text{再現率} = \frac{|G \cap R|}{|G|}$$

検証結果を表 2 に示すに示す。

表 2　処理精度結果

処理精度	Matrix Profile	Basket	Smartphone	Mt.Tsukuba
提案手法の適合率	100%	100%	100%	100%
提案手法の再現率	100%	100%	100%	100%
LSH 手法の適合率	100%	100%	100%	100%
LSH 手法の再現率	100%	100%	100%	100%
DFT 手法の適合率	100%	97%	91%	92%
DFT 手法の再現率	100%	100%	100%	100%

検証結果から，提案手法，LSH 手法ともに 100%の精度で検索を行えることがわかる．これは，グループ検索の後に元の時系列データへの復元を行っているためである．一方，DFT 手法では適合率が下がってしまう場合があった．これは，文献 [3] にも述べられているように，類似部分シーケンスを見逃すことはないが，多く検索してしまうことはあるという性質によるものだと考えられる．

5.3　提案手法の問合せ処理時間と処理精度の変化

本節では，提案手法のクエリ長と部分シーケンス長の関係による，問合せ処理時間と処理精度の変化について検証を行う．提案手法の余剰部分の除外や平均値で埋めるという操作が，問合せ処理時間や処理精度にどのような影響をもたらすのかを検証する．具体的には，実データセットに対して提案手法を用いて検索を行う際に，クエリ長を部分シーケンス長の 0.6 倍から 1.3 倍まで変化させて検証を行う．

検証結果を図 6,7 に示す．図 6 は問合せ処理時間の変化を，

図 7 は処理精度の変化を表している．また，図 6,7 の x 軸はクエリ長を表している．例えば，$x = 0.8$ のときは，クエリ長が部分シーケンス長の 0.8 倍であることを表している．図 6 の y 軸は問合せ処理時間（秒）を，図 7 の y 軸は処理精度（%）を表している．

図 6 より，クエリ長は問合せ処理時間に大きな影響を与えないことがわかる．一方，図 7 より処理精度には大きな影響を与えることがわかる．特に，クエリ長が部分シーケンス長より短いときは，余剰部分を平均値で埋める操作を行っているからだと考えられる．しかし，クエリ長が部分シーケンス長より長いときは高い処理精度を出せている．これは，余剰部分を除外する操作は処理精度に影響を与えないことを表している．

6 結　　論

本研究ではクエリ長に依存しない多次元時系列データに対する高速な類似部分シーケンス問合せ手法を提案した．実データを用いた評価実験において，比較手法に対し提案手法が有効であることを確認できた．今後の課題として，クエリ長が部分シーケンス長よりも短い場合の処理精度の改善が挙げられる．この部分を改善することで，より多くの多次元時系列データに対して提案手法を用いることが可能になると考えられる．

謝　　辞

本研究の一部は，JST さきがけ (JPMJPR2033) ならびに JSPS 科研費 (JP22K17894) の支援を受けたものである．

文　　献

[1] Ryuichi Yagi and Hiroaki Shiokawa. Fast top-k similar sequence search on dna databases. In *Information Integration and Web Intelligence: 24th International Conference, iiWAS 2022, Virtual Event, November 28–30, 2022, Proceedings*, pp. 145–150. Springer, 2022.

[2] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pp. 2–11, 2003.

[3] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. *Acm Sigmod Record*, Vol. 23, No. 2, pp. 419–429, 1994.

[4] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th international conference on data mining (ICDM)*, pp. 1317–1322. Ieee, 2016.

[5] Gineke A Ten Holt, Marcel JT Reinders, and Emile A Hendriks. Multi-dimensional dynamic time warping for gesture recognition. In *Thirteenth annual conference of the Advanced School for Computing and Imaging*, Vol. 300, p. 1, 2007.

[6] Yuma Yasuda and Hiroaki Shiokawa. Efficient similarity searches for multivariate time series: A hash-based approach. In *International Conference on Information Integration and Web Intelligence*, pp. 478–490. Springer, 2023.

[7] 天方大地, 原隆浩. 相関時系列データ集合の計算のための高速アルゴリズム. In *IEICE Conferences Archives*. The Institute of Electronics, Information and Communication Engineers, 2017.

[8] 古賀久志. ハッシュを用いた類似検索技術とその応用. 電子情報通信学会 基礎・境界ソサイエティ Fundamentals Review, Vol. 7, No. 3, pp. 256–268, 2014.

[9] Chin-Chia Michael Yeh, Nickolas Kavantzas, and Eamonn Keogh. Matrix profile vi: Meaningful multidimensional motif discovery. In *2017 IEEE international conference on data mining (ICDM)*, pp. 565–574. IEEE, 2017.

[10] Abdulwahed Salam and Abdelaaziz El Hibaoui. Comparison of Machine Learning Algorithms for the Power Consumption Prediction : - Case Study of Tetouan city –. In *Proceedings of the 6th International Renewable and Sustainable Energy Conference (IRSEC 2018)*, pp. 1–5, 2018.

[11] Paolo Barsocchi, Antonino Crivello, Davide La Rosa, and Filippo Palumbo. A Multisource and Multivariate Dataset for Indoor Localization Methods based on WLAN and Geo-Magnetic Field Fingerprinting. In *Proceedings of the 2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN 2016)*, pp. 1–8, 2016.

[12] University of Tsukuba Center for Computational Sciences. Mt. tsukuba project, 2023. (July 25th, 2023 Accessed).

NexmonによるCSIベースのスマートデバイス状態推定

原田　海斗†　　寺本　京祐††　　野村裕一郎†††　　峰野　博史††††

† 静岡大学情報学部　〒 432–8011 静岡県浜松市中区城北 3 丁目 5-1
†† 静岡大学大学院総合科学技術研究科情報学専攻　〒 432–8011 静岡県浜松市中区城北 3 丁目 5-1
††† 静岡大学学術院情報学領域　〒 432–8011 静岡県浜松市中区城北 3 丁目 5-1
†††† 静岡大学グリーン科学技術研究所　〒 432–8011 静岡県浜松市中区城北 3 丁目 5-1
E-mail: †{harada.kaito.20,teramoto.kyosuke.18}@shizuoka.ac.jp,
††{nomura.yuhichiro,mineno}@inf.shizuoka.ac.jp

あらまし　現在のスマートデバイスは，利用者自身でも動作状態の把握が難しく，不正な通信や動作が行われている場合に気づく術がない．そこで，スマートデバイスの状態 (アプリケーションの動作状況，操作内容など) を推定するシステムの実現を目指している．従来の研究では，スマートデバイスの状態推定手法として様々なアプローチが検討されているが，汎用性や導入コストの面で懸念が残る．本研究では，Wi-Fi 電波の通信媒体波及時におけるチャネル状態情報 (Channel State Information; CSI) を，CSI 収集用ファームウェアパッチである Nexmon を用いて収集・分析し，スマートデバイス状態推定を行う手法を提案する．基礎評価として，6 種類のアプリケーションに対して 8 種類の時系列モデルでの試行の結果，最大 87.6%の精度でアプリケーション推定が可能なことを確認した．また，分類アプリケーションの組み合わせによる精度変化を検証し，最大 100%で推定できることを明らかにした．
キーワード　Wi-Fi，CSI，Nexmon，時系列データ，スマートデバイス，アプリケーション推定

1　はじめに

近年，スマートフォンやパソコンなど多機能 IoT デバイス (スマートデバイス) が広く普及し，その活用シーンは多岐にわたる．世界のスマートデバイスの普及台数は年々増加しており，2025 年には約 440 億台に達すると予測されている．スマートデバイスの増加に伴って，攻撃者によるスマートデバイスに対しての不正操作や情報漏洩が懸念される．しかし，現在のスマートデバイスは，利用者自身でも動作状態の把握が難しく，不正な通信や動作が行われている場合に気づく術がない．

そこで，スマートデバイスにおけるアプリケーションの動作状況を推定するシステムの実現を目的としている．このシステムを活用することで，利用者自身がスマートデバイスの動作状況を把握し，第三者による不正な操作や情報漏洩を未然に防ぐことができる．また，スマートデバイスの使用状況を管理するシステムに組み込むことによって，より頑健なスマートデバイス管理システムの実現も期待できる．

従来の研究では，スマートデバイスの通信トラヒックから得られる統計情報を用いる手法 [1] や，状態遷移モデルと IoT センサ情報を用いたエッジコンピューティングによる手法 [2] が検討されている．しかし，通信トラヒックを用いた手法では，情報量に限界がありスマートデバイスの状態を詳細に捉えることが難しい．また，状態遷移モデルと IoT センサ情報を用いた手法では，状態遷移モデルの汎用性，導入コストの高さが懸念される．

本研究では，Wi-Fi 電波の通信媒体波及時における CSI を，CSI 収集用ファームウェアパッチである Nexmon [3] を用いて収集・分析し，スマートデバイス状態推定を行う手法を提案する．

2　関連研究

2.1　通信トラヒック分析による複数のスマートデバイスにおける機能推定手法の評価 [1]

ここでは，複数のスマートデバイスが有線・無線でルータに接続される環境を想定している．接続されるデバイス全てが，単一のエッジルータを介してインターネット接続を行う環境で通信トラヒックを収集している．また，通信トラヒックから 32 個の特徴量を算出し，RandomForest へ入力することで推定を行う手法が提案されている．評価データとして，8 機種のスマートデバイスで 8 種類の機能を実行した際の通信トラヒックを収集している．結果として，スマートデバイスの機種と機能の 16 通りに対して，88%の精度で分類できることを明らかにした．

しかし，通信トラヒックを用いた手法では，機種や機能による通信トラヒックの差異が小さく，スマートデバイスの状態を詳細に捉えることが難しい．これは，通信トラヒックによる特徴量が持つ情報量が少ないことが起因している．そのため，より詳細なスマートデバイスの状態推定を行うためには，通信トラヒック以外の情報を用いる必要がある．

2.2　状態遷移モデルと IoT センサによるエッジコンピューティングを用いたスマートデバイスの異常検知 [2]

実際の家庭環境に設置されているスマートデバイスの使用状

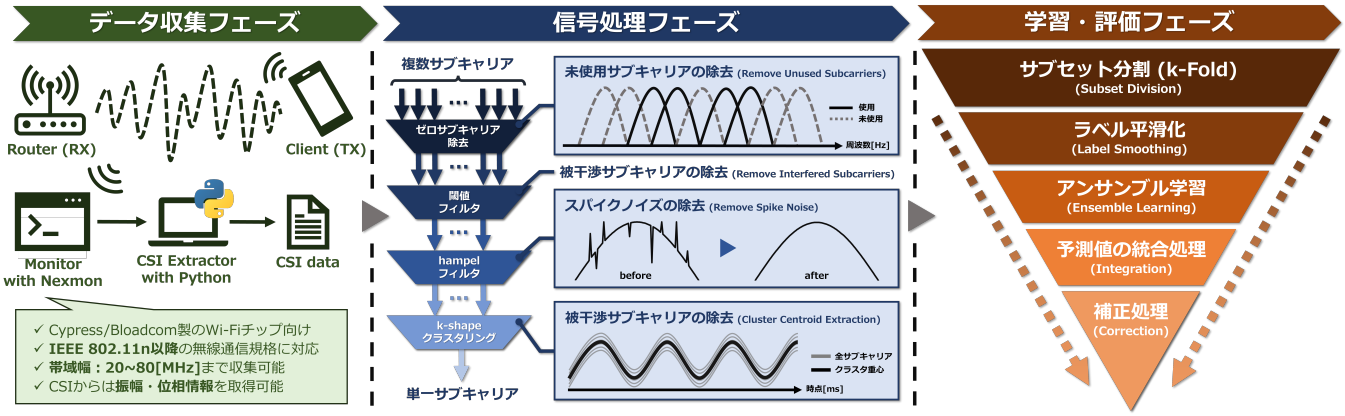


図 1　提案手法：概要図

況を，各デバイスに対応する IoT センサ値とエッジコンピュータを用いて収集している．また，家庭の活動状況を考慮した状態遷移モデルを，IoT センサ値を用いて構築し，不正操作を検出する手法が提案されている．

結果として，誤検出率：20.1%未満，検出率：72.3%でスマートデバイスの状態推定が可能なことを明らかにした．

しかし，この手法では各スマートデバイスに対して状態遷移モデルを構築する必要があり，汎用性の観点で懸念が残る．また，スマートデバイスの操作状況を収集するために，各デバイスそれぞれに IoT センサと，エッジコンピュータが必要であるため，導入コストの高さも懸念される．そのため，より汎用性が高く，導入コストの低いスマートデバイスの状態推定手法が必要である．

3　提 案 手 法

Wi-Fi 電波の通信媒体波及時における CSI を，CSI 収集用ファームウェアパッチである Nexmon を用いて収集・分析し，スマートデバイス状態推定を行う手法を提案する．CSI は，通信技術分野において広く使用される通信路の状態情報である．具体的には，Wi-Fi 電波が送受信デバイス間で伝播する際の電波散乱，フェージング，到来距離・角度による電力減衰などの複合的な要因を考慮したチャネル状態を表し，振幅・位相情報を持つ．CSI は，本来は通信品質の評価や通信路推定などに使用されるが，本研究では，スマートデバイスの動作状況を推定するための特徴量として利用する．CSI は通信トラヒックや RSSI(Received Signal Strength Indicator) と比較して取得できる情報が多く，スマートデバイスの状態を詳細に捉えることができる．CSI を収集するソフトウェアとして Nexmon を使用する．Nexmon を使用することで，CSI 収集不可能な低コストデバイスでも CSI 収集可能となる．

図 1 に提案手法の概要図を示す．提案手法は，「データ収集フェーズ」「信号処理フェーズ」「学習・評価フェーズ」の 3 つのフェーズで構成される．

3.1　データ収集フェーズ

「データ収集フェーズ」では，CSI を Nexmon を適用したデバイスを用いて取得し，Python ベースの CSI 解析プログラムを用いて，各サブキャリアの振幅・位相情報を取得する．このとき，スマートデバイスが Wi-Fi ルータのクライアントとして IEEE802.11n 規格以降の通信方式 (OFDM や MIMO などのマルチキャリア・アンテナ通信) かつ，帯域：2.4/5[GHz] で帯域幅：20/40/80[MHz] で通信を行う状況を想定する．

Nexmon 適用デバイスは，指定されたチャネルと帯域幅に該当する通信を観測するため，スマートデバイスが接続しているネットワークに接続する必要はない．そのため，多様なタスクに対して適用可能なデータ収集手法であり，システム全体として汎用性が高くなる．また，新たに導入するデバイスは Nexmon を適用するデバイスのみであるため，従来手法と比較して導入コストが低い．

3.2　信号処理フェーズ

「信号処理フェーズ」では，データ収集フェーズで取得した CSI データの前処理を行う．前処理として，ゼロサブキャリア除去，閾値フィルタ，hampel フィルタ，k-shape クラスタリングを適用する．

ゼロサブキャリア除去は，CSI データの中で振幅が 0 のサブキャリアを除去する処理である．IEEE802.11n 規格以降で使用されるマルチキャリア通信では他チャネルとの干渉を抑制するため，一部のサブキャリアをデータ通信に使用しない工夫がされている [4]．そのため，それらのサブキャリア情報の除去をすることで通信に使用されたサブキャリア情報のみを抽出する．

閾値フィルタでは，振幅情報が一定値を超えるサブキャリアを除去するフィルタである．ゼロサブキャリアに隣接するサブキャリアは他チャネルからの干渉を受けやすく，ノイズが多く含まれるため，学習には不向きである．具体的には，ゼロサブキャリアを除く k 個のサブキャリア $S_k \ni s_k$ に対して式 1 を適用し，閾値 α を超えるサブキャリアを除去する．

$$threshold(S_k) = \begin{cases} S_k & if\ s_k < \alpha \\ None & otherwise \end{cases} \quad (\alpha > 0) \qquad (1)$$

hampel フィルタ [5] では，選定されたサブキャリア集合の各要素に対してノイズ処理を行う．Nexmon によって収取された

CSI データにはスパイクノイズが多く含まれている．そのため，外れ値検知アルゴリズムである hampel フィルタはノイズ除去として有効に作用する．具体的には，単一サブキャリアに対して，中央値 $\widetilde{X_i}$，標準偏差 σ_i のスライディングウィンドウ $X_i \ni x_i$ を作成する．その後，全スライディングウィンドウに式 2 を適用し，$\beta\sigma_i$ を超える要素 x_i を中央値 $\widetilde{X_i}$ に置換する．

$$hampel(X_i) = \begin{cases} \widetilde{X_i} & if\ |x_i - \widetilde{X_i}| > \beta\sigma_i \\ x_i & otherwise \end{cases} \quad (\beta > 0) \quad (2)$$

k-shape クラスタリングでは，形状ベースの時系列クラスタリング手法である k-shape [6] を適用し，全サブキャリアクラスタの重心信号を抽出する．ここで，サブキャリアは変動幅が異なる程度の差異しか見られず，相関係数が非常に高い．そのため，全サブキャリア情報をすべて使用すると学習コストが高くなり，共線性による過学習に陥る可能性がある．時系列クラスタリングによって得られる重心信号を代表値として学習に使用することで，学習コストの低減を図る．

3.3 学習・評価フェーズ

「学習・評価フェーズ」では，前処理を行った CSI データを入力として時系列モデルによる学習・評価を行う．具体的には，ラベル平滑化 [7] を施した学習データに対して，k-Fold アンサンブル学習モデルを構築し，予測値の統合・補正を行うことで最終予測値を得る．ラベル平滑化は，K クラスの One-Hot エンコーディングされた目的変数 $y_k = [y_1, ..., y_K]$ に対して，ノイズ γ を加える手法 (式 3) である．これにより，モデルの過学習を抑制し，類似した特徴量に対する分類精度，汎化性能が向上する．

$$smoothing(y_i) = y_i(1-\gamma) + \frac{1}{K}\gamma \quad (0 < \gamma < 1.0) \quad (3)$$

k-Fold アンサンブル学習時において使用するサブセットはスライディングウィンドウ形式で作成され，目的変数の分布はランダムである．また，スライディングウィンドウは，ウィンドウサイズ (入力時点数) が可変かつ，複数の目的変数が含まれないように作成する．分割された k 個のサブセットそれぞれに対して，同一パラメータ・アーキテクチャの学習モデルを構築する．得られる k 個の予測値は，相加平均によって統合する．また，統合された予測値に対して，過去の予測値から妥当性を評価し，適切な値に補正することで最終予測値を得る．

4 基礎評価

4.1 データ収集フェーズ

図 2 に，基礎評価における CSI データの収集環境を示す．スマートデバイスとして iPhone12，Wi-Fi ルータとして WSR-1800AX4P-WH を使用した．また，スマートデバイス上で動作するアプリケーションは，TikTok，LINE，パズル&ドラゴンズ (Puzzle)，雀魂 (Mahjong)，YouTube，コミックシーモア (Comic) の 6 種類とした．Nexmon を適用する CSI 収集デバイスとして，安価かつ入手が容易なことから RaspberryPi4B を使用した．通信条件は，スマートデバイスと Wi-Fi ルータ間の距離を 1.5[m] で配置し，通信規格：IEEE802.11n の帯域：2.4[GHz]，帯域幅：20[MHz] として通信を行った．収集条件は，サンプリングレート：5.0[Packet/s] として収集を行った．

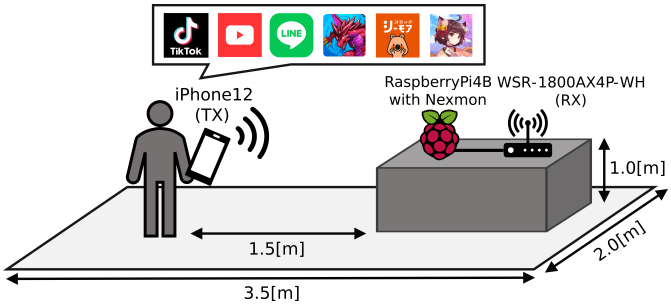


図 2　基礎評価：CSI データ収集環境

表 1　実 行 環 境

CPU	Intel Core i9-13900KF
GPU	GeForce RTX 3090
OS	Ubuntu 22.04.2 LTS
Software	Python3.9, tensorflow2.12, CUDA11.8

4.2 信号処理フェーズ

収集した CSI は，64 個のサブキャリア情報で構成される．IEEE802.11n 規格では，8 つのサブキャリアはデータ搬送には使用されないため，ゼロサブキャリアとして除去を行った．また，式 1 の閾値フィルタ ($\alpha = 3000$) を適用し，振幅情報が一定値を超えるサブキャリアの除去を行った．また，抽出された各サブキャリアに対して，式 2 の hampel フィルタ ($\beta = 2.0$) によるスパイクノイズの除去を行い，k-shape を用いて全サブキャリアの代表値を抽出した．

4.3 学習・評価フェーズ

学習・評価フェーズでは，表 1 に示す実行環境下で学習を行った．目的変数に対して式 3 のラベル平滑化処理 ($\gamma = 0.1$) を施した．また，k-Fold アンサンブル学習モデルに入力するサブセットの分割数は 3 とし，相加平均によって予測値の統合を行った．補正処理として，分類タスクにおいて誤分類した箇所がスパイク状に出現することから，式 2 の hampel フィルタ ($\beta = 1.5$) を使用した．

4.3.1 時系列モデルと入力時点数の選定

k-Fold アンサンブル学習モデルとして，8 種類の時系列モデル (RandomForest，TCN，1D-CNN，LSTM-FCN，Transformer，1D-ResNet，1D-DenseNet) を試行した．

表 2 に，各時系列モデルによる精度比較を示す．各時系列モデルに対して，入力時点数を 200，250，300，350，400，450[Packet] と変化させ，各学習モデルで得られる最大精度を記録した．各時系列モデルにおける準最大精度を下線，最大精度を太字で表記した．ラベル数に極端な偏りがないことから，評価指標は Accuracy[%] とした．最良結果として，学習

表 2　時系列モデルと入力時点数による精度比較 (Accuracy[%])

入力時点数	時系列モデル							
	RandomForest	TCN	1D-CNN	LSTM-FCN	Transformer	1D-ResNet	1D-DenseNet	LSTM
200	**67.9**	65.9	66.7	63.4	56.7	**72.1**	57.5	36.7
250	64.0	72.3	70.9	62.3	60.9	67.4	75.4	**51.6**
300	63.5	84.0	70.2	68.4	59.7	71.1	76.5	41.2
350	60.8	**87.6**	71.1	**68.5**	61.6	63.9	73.9	40.9
400	57.1	71.7	72.0	63.1	61.8	68.6	72.1	47.1
450	58.5	60.4	**76.0**	68.0	**62.4**	66.3	**80.9**	39.3

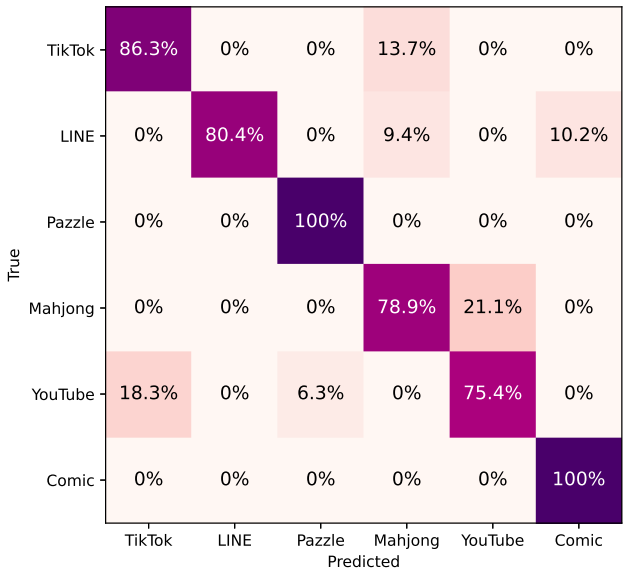


図 3　最良結果における分類結果 (混同行列)

モデル：TCN(Temporal Convolutional Network)，入力時点数：350[Packet](≃ 70[s]) とした時の Accuracy：87.6[%](F 値：0.871) を確認した．

図 3 に，表 2 で最良結果を確認した条件下での分類結果 (混同行列) を示す．分類難易度がアプリケーションによって異なり，誤分類しやすい組み合わせが存在することが確認できる．

4. 3. 2　入力時点数による精度比較

本節の第 1 項で最良結果を確認した学習モデル：TCN に対して，入力時点数を変化させた場合の精度比較を行った．入力時点数を 100〜500[Packet]，hampel フィルタのパラメータを $\beta = 1.0$〜3.0 として試行した．

図 4 に，各入力時点数ごとに得られた最大精度の推移を示す．

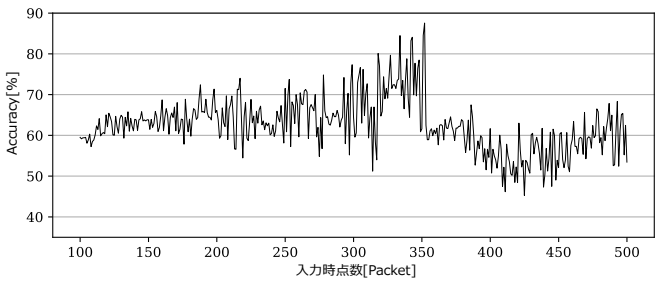


図 4　入力時点数による精度比較

表 3　アプリケーションによる精度比較 (● 分類対象，○ 非分類対象)

アプリケーションの組み合わせ						Acc.[%]
TikTok	LINE	Puzzle	Mahjong	YouTube	Comic	
●	●	●	●	●	○	94.4
●	●	●	●	○	●	71.9
●	●	●	○	●	●	90.7
●	●	○	●	●	●	89.1
●	○	●	●	●	●	94.1
○	●	●	●	●	●	89.8
●	●	●	●	○	○	66.5
●	●	●	○	○	●	66.4
●	●	○	○	●	●	91.8
●	○	○	●	●	●	90.3
○	○	●	●	●	●	95.4
○	●	●	●	●	○	**100.0**

入力時点数が増加するに伴い，分類精度の向上する傾向が確認できる．しかし，それと同時に入力時点数が 350[Packet] を超えたあたりで急激な精度低下がみられる．

4. 3. 3　アプリケーションによる精度比較

本節の第 1 項で最良結果を確認した学習モデル：TCN に対して，アプリケーションの組み合わせによる精度比較を行った．

表 3 に，分類対象とするアプリケーションの組み合わせに対する精度比較を示す．分類対象とするアプリケーションの組み合わせによって，分類精度に差異が生じることが確認できる．

5　考　　察

はじめに，図 3 における分類結果について考察する．“Puzzle” と “Comic” はどのアプリケーションとも誤分類することなく，適切に分類ができている．しかし，“Mahjong” と “YouTube” は分類精度が 80%を下回っており，基礎評価で対象としたアプリケーションの中で分類難易度が高いと言える．“Mahjong” は “YouTube” との誤分類が目立っているが，これは両者の操作方法 (横持ち，操作時の指の動きなど) が類似することに起因していると考えられる．また，“YouTube” は “TikTok” との誤分類が目立っており，これは両者の内部特性 (ストリーミング，バッファリングなど) が類似することに起因していると考えられる．

次に，図 4 における入力時点数による精度比較ついて考察する．一般的に，入力時点数が増加すると入力データの情報量が増加し，学習モデルの分類精度は増加する．しかし，特定の入力時点数 (=350[Packet]) を超えると，各アプリケーションが

持つ特徴量の差異が小さくなり，誤分類を引き起こしている可能性がある．

最後に，表 3 におけるアプリケーションによる精度比較について考察する．一般的に，分類クラス数の減少に伴って，分類精度は増加する．しかし，分類対象とするアプリケーション数が減少した場合に，分類精度が低下する組み合わせが存在している．提案手法における学習・評価フェーズでは，モデルの汎化性能の向上を目的としたラベル平滑化処理行っている．ラベル平滑化処理は，分類クラス内に類似する特徴量を持つ組み合わせが存在する場合に有効な手法である．しかし，分類アプリケーション内に特徴量が類似する組み合わせが存在しない場合，ラベル平滑化はモデルの性能を低下させてしまう可能性がある．このことから，分類対象とするアプリケーションの組み合わせによっては類似した特徴量を持つアプリケーションが少ないため，分類精度が低下していると考えられる．

6 議　　論

はじめに，スマートデバイス状態推定において使用する CSI の特徴量に影響を与える要因について議論する．スマートデバイス操作によって発生するトラヒックやトランザクション，操作方法・位置・環境などが挙げられる．それらを明かにするためには，トラヒックやトランザクションを可視化し，CSI 特徴量との関連性を検証する必要がある．また，非操作時と操作時の場合と操作位置に動的な変化を加えた場合の CSI 特徴量をそれぞれ分析する必要がある．

次に，基礎評価で対象としたスマートデバイス，およびアプリケーション以外に対する有効性について議論する．スマートデバイスに関しては，通信を行う無線通信規格，帯域幅などの通信条件を一致させた場合は，他のスマートデバイスにも適用可能であると考えられる．しかし，アプリケーションに関しては，同様のアプリケーションであってもバージョンや設定による差異が考えられる．それらを解明するには，異なるスマートデバイスと，異なるバージョンや設定のアプリケーションの組み合わせによる CSI 特徴量の変化を検証する必要がある．

最後に，複数人のスマートデバイスを対象とした場合の有効性について議論する．複数人が同時に，通信条件の異なるスマートデバイスを操作する場合，単一の Nexmon 適用デバイスで複数の CSI を取得することはできない．そのため，通信条件が異なる場合は，複数の Nexmon 適用デバイスを用いて，複数人のスマートデバイス状態推定を行う必要がある．通信条件が同一であっても，複数人を対象とする場合は，CSI 特徴量を分離する必要がある．

7 おわりに

Nexmon による CSI を用いたスマートデバイス状態推定手法を提案し，基礎評価を行った．8 種類の時系列モデルを試行した結果，学習モデル：TCN によって最大分類精度 87.6%を確認した．また，入力時点数による精度比較を行うことで，スマートデバイス状態推定における最適な入力時点数を明らかにした．さらには，アプリケーションの組み合わせによる精度比較を行うことで，アプリケーションによる分類難易度を検証した．

本研究のスマートデバイス状態推定手法は CSI データを用いるため，通信トラヒックを使用する従来手法 [1] と比較して，取得できる情報が多く，スマートデバイスの状態を詳細に捉えることができる．また，複数の IoT センサとエッジコンピュータを用いる従来手法 [2] と比較して，新たに導入するデバイスが少なく，導入コストが低いといえる．加えて，ドメイン知識に基づいた状態遷移モデルを構築する必要がないため，汎用性が高いといえる．

今後の展望として，スマートデバイス状態推定モデルの精度向上，結果に起因する内外的要因 (個人差や環境，動作アプリケーションの数・種類など) についての検証を進める．また，時系列向けの半教師あり学習 [8] や，データ拡張手法 [9] と組み合わせることで，学習コストが低く，汎用性の高い学習を検討する．最終的には，本研究の成果を活用し，スマートデバイスの動作状態を推定するシステムの実現を目指す．

謝　　辞

本研究の一部は，静岡大学グリーン科学研究所プロジェクト研究支援 (23205) を受けたものである．

文　　献

[1] 祐一服部, 豊荒川, 創造井上. 通信トラヒック分析による複数の iot デバイスにおける機能推定手法の評価. マルチメディア，分散，協調とモバイルシンポジウム 2022 論文集, Vol. 2022, pp. 655–661, 2022.

[2] 田中雅弘, 山内雅明, 大下裕一, 村田正幸, 上田健介, 加藤嘉明. 家庭活動の状況推定を用いたスマートホームネットワークの異常検出手法. IEICE Technical Report; 信学技報, Vol. 119, No. 461, pp. 219–224, 2020.

[3] Francesco Gringoli, Matthias Schulz, Jakob Link, and Matthias Hollick. Free your csi: A channel state information extraction platform for modern wi-fi chipsets. pp. 21–28, 2019.

[4] Defeng David Huang and Khaled Ben Letaief. Carrier frequency offset estimation for ofdm systems using null subcarriers. *IEEE Transactions on Communications*, Vol. 54, pp. 813–823, 2006.

[5] Ronald K. Pearson, Yrjö Neuvo, Jaakko Astola, and Moncef Gabbouj. The class of generalized hampel filters. In *EUSIPCO*, pp. 2501–2505, 2015.

[6] John Paparrizos and Luis Gravano. k-shape: Efficient and accurate clustering of time series. *SIGMOD Rec.*, Vol. 45, pp. 69–76, 2016.

[7] Rafael Müller, Simon Kornblith, and Geoffrey Hinton. When does label smoothing help?, 2020.

[8] Haoyi Fan, Fengbin Zhang, Ruidong Wang, Xunhua Huang, and Zuoyong Li. Semi-supervised time series classification by temporal relation prediction. In *ICASSP*, pp. 3545–3549, 2021.

[9] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. Data augmentation for time series classification using convolutional neural networks, 2016.

LSiX: ストリームデータに関する複数連続的集約

川上　隼† Bou Savong† 天笠　俊之††

† 筑波大学システム情報工学研究群　〒 305-8573 茨城県つくば市天王台 1-1-1
†† 筑波大学計算科学研究センター　〒 305-8577 茨城県つくば市天王台 1-1-1
E-mail: †s2320583@u.tsukuba.ac.jp, ††{savong-hashimoto,amagasa}@cs.tsukuba.ac.jp

あらまし　ストリームデータ分析の需要が増加するにつれて、ストリーム処理エンジンはいくつかの技術的課題に対処する必要がある。連続的な window の集約はさまざまなアプリケーションでデータ分析のパイプラインの一部に使われており、高い入力レートのストリームデータにおいて大量のクエリを処理するときにスケーラビリティの問題に直面する。この問題に対処するために LSiX(Longest-shortest-window based indexing) というストリームデータに対して複数クエリを効率的に集約するアルゴリズムを提案する。この手法は、複数のクエリのうち最長と最短の window をもとにした 2 つの配列を利用して、全てのクエリを二つの配列から共有した値から計算することにより、各クエリを最大 2 回の集約操作で計算することを可能にした。実験により、最新手法である MCQA を含む比較手法より、3.5 倍以上高速であることが示された。

キーワード　ストリームデータ処理、すべり窓集約

1. はじめに

近年、ネットワーク監視や経済など、さまざまな領域で連続的なデータストリームが普及し、データストリー処理への関心が強くなっており、データストリーム管理システム (DSMS) [1] における効率的でリアリタイムに近い処理の必要性が高まっている。また、時系列データ [3] の sum, max, min, average などの集約計算に対する需要も大幅に増加している。例えば、株式投資家は連続的な株価に頼るだけでなく、彼ら独自のリアルタイムな集約 [10] の結果も必要としている。このような場合、ユーザーはデータストリームにおける最新のデータに関して主に興味がある。

データストリームを集約するには、一般的に sliding window aggregation(SWAG) [2], $Q[w, s]$, を使用する。SWAG は、新しい s 個のレコードが到着するたびに window 内の w 個の最新のレコードを集約する手法である。SWAG には time-based と count-based の 2 種類が存在する。前者は window を時間単位で区切り、後者は window をレコードの数によって区切る。多くのアプリケーションで、DSMS はデータストリームに対して複数のクエリを処理する必要があり、クエリの数は多く、数千となることもある。その上、DSMS はそのようなクエリを遅延時間を少なく処理する必要がある。しかし。このようなデータストリームを従来の処理する場合、スケーラビリティの問題に直面する。

この問題を対処するために SWAG のための効率的なアルゴリズムが必要となる。単一クエリのための SWAG の研究 [4] は多くあるが、複数クエリを処理するとき、クエリの数に比例して slide 毎のレイテンシーが増加する。このオーバーヘッドにより、これらのアルゴリズムは何千ものクエリを同時に処理することは現実的ではない。

MCQA [9] は複数クエリを処理するための最新のアルゴリズムである。しかし、MCQA は window サイズと slide サイズの比が大きい時や異なる window サイズの差が大きい時などで処理時間が大幅に増えてしまうという問題がある。

この問題に対処するために、この論文ではデータストリームに対して、複数クエリの SWAG を効率的に処理するためのアルゴリズムである LSiX(longest-shortest-window-based-indexing) を提案する。LSiX はクエリにおける最長と最短の window サイズを使用して二つの配列 Lo, Sh を作成し、それらの集約結果を全てのクエリに共有することで、安定性を確保し、処理時間を大幅に短縮する。実験から最新の手法である MCQA と比べて少なくても 3. 5 倍速く、複数のクエリを処理できるように拡張された L-BiX [4] よりも少なくても 3 倍速く処理することがわかった。

2. 前提知識

このセクションでは、この後の議論に必要な前提知識を提供する。

2.1 Sliding Window Aggregatin と部分集約

データストリームが与えられた場合、SWAG は window 内のレコードに対する集約を連続的に計算する。クエリは window サイズ、slide サイズ、集約関数によって定義される。

複数の SWAG クエリを同時に処理するときに、全ての window を最初から集約するのは非効率である。第一に同じクエリの slide の前後二つの window は互いに重なる場合があるため、重なった部分は再利用することができる。第二に異なる SWAG クエリによって定義された window も互いに重なる部分があるため、一部の結果を再利用することができる。

一つの手法として window を分割して、部分集約を計算す

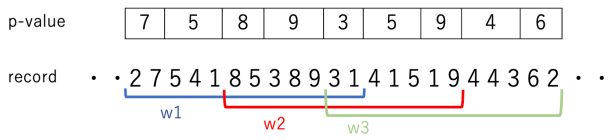


図 1: max を求めるクエリ:$Q[12,5]$ の部分集約の例.

る方法がある。関連するパーティションによって計算された部分集約を使用することで各 SWAG クエリの最終結果を求めることができる。window を分割するための技術には、Panes [8], Pairs [7], Cutty [5] がある。クエリ $Q[w,s]$ が与えられた時、Pairs は $f1 = w\%s, f2 = S - f1$ で求められる 2 つのパラメータを使って window を分割する。例えば、$Q[12,5]$ というクエリで集約関数が max の SWAG を考える（図 1）。部分集約を使わない場合は slide するごとに window 全体を集約するため、11 回の集約操作が必要となる、それに対して Pairs を使った場合、$f1 = 12\%5 = 2, f2 = 5 - 2 = 3$ となり、w1 の集約結果は 9=max(7,5,8,9,3）と求まる。部分集約を使うことで、slide ごとの集約は、f1 と f2 を求めるのに 3 回と最終結果を求めるのに 4 回の計 7 回の集約操作で最終結果を求めることができる。

2.2 集約関数

集約関数 [6] は複数の値を集約することで、生のデータから高水準な情報を抽出するために頻繁に使用される。多くの集約関数が存在するが、それらは以下の 3 つに分類される。

- Distributive: 分散して計算できる集約関数。例）sum, max, count など。その関数で求めた部分集約の値を使って全体の集約を求めることができる。
- Algebraic: 固定された有限引数を持つ代数関数によって計算できる集約関数。例）平均、分散など。平均は distributive な関数である sum を count で割ることで求めることができる。
- Holistic: 計算するための固定された有限引数を持つ代数関数がない集約関数。例）中央値、四分位偏差など。つまり、上記の二つに属さない関数は Holistic な関数となる。

本論文では、Distributive と Algebraic な関数を扱うが、Holistic な関数への対処は今後の課題となる。

3. 関連手法

効率的に SWAG を計算するための手法は多く存在する。まずは最も関連した研究に焦点を当てる。

L-BiX [4] は単一の SWAG クエリを効率的に求めるための手法である。最新のレコードから最古のレコードまでを順に部分集約値を求める後方集約と最古のレコードから最新のレコードまでを順に部分集約値を求める前方集約を結合することで効率的に集約計算をする手法となる。L-BiX は平均して 3 回の集約操作で結果を求めることができる。時間計算量は O(1) となり、現在データストリームに対して単一クエリを処理する最も効率的な手法である。

Two-Stack [12] と FlatFIT [11] も単一の SWAG クエリを処理する手法で、部分集約値とポインタを格納する 2 つの配列を使用し、ポインタを介して必要な部分集約地のインデックスを取得し、最終結果を得る。FlatFIT は window 内の部分集約の数が nc の時、平均して $log(nc)$ 回の集約操作で結果を求めることができる。Two-Stacks は時間計算量が O(1) であるが、L-BiX よりも slide ごとの更新回数が多く、より大きなメモリを必要とする。

一方、複数の SWAG クエリを同時に処理する必要が多く、上記の手法は単一のクエリ処理に最適化されているため、複数のクエリを効率的に処理することができない。MCQA [9] はこの問題に対応した手法となる。MCQA は以下の手順で特定のストリームに対する複数のクエリを同時に集約することができる：(1) 複数のクエリの window のサイズを昇順に並び替える;(2)slide サイズの集約結果を保持する;(3) それを再利用して小さい window サイズのクエリから順に集約を実行する。また、大きな window サイズを持つクエリを集約する際にはそれより小さい window サイズのクエリの結果を保持し再利用することで効率的な集約を行う。

しかし、MCQS の欠点として window サイズと slide サイズによって計算量が変化し、特定のパラメータの組み合わせで実行時間が長くなるという不安定さがある。以下の 3 つの状況で実行時間が長くなる:(1) 連続する 2 つの window サイズの差が大きい,(2) 最小の window サイズと slide サイズの比率が大きい,(3) 全ての window サイズが互いに倍数でない。例は図 2 に示す。レコード:3(最も右のレコード) が到着した時、$w1$ は単純に最新の 5 個のレコードを集約することによって計算される。$w2$ は $w1$ を再利用することで 3 回の集約操作 $(w2[0] = max(w1[0], 4, 9, 6)$ で求めることができ, $w3$ は $w2$ を再利用することで 4 回の集約操作 $(w3 = max(w2[0], 5, 3, 8, 9)$ で求めることができる。しかし、$w2$ と $w3$ の差が大きくなると集約操作の回数も増加する。

n 個の window サイズ:$W_1, W_2, ..., W_n (W_1 < W_2 < ... < W_n)$ と slide サイズ:Sl が与えられた時、MCQA で slide ごとに全ての結果を計算するのに必要な集約操作の回数は $\sum_{i=2}^{n}(\lfloor \frac{W_i}{W_{i-1}} \rfloor + W_i\%W_{i-1}) + \lfloor \frac{W_1}{Sl} \rfloor + W_1\%Sl$ となり、必要なスペースは $\lceil \frac{W_{max}}{Sl} \rceil \times n$ となる。また、単一のクエリを求める最新の手法である L-BiX の計算量は $3n$ でスペースは $\sum_{i=1}^{n}(\frac{W_i}{Sl})$ となる。必要な時間とスペースはテーブル 1 にまとめた。L-BiX の必要な時間は n に依存する。しかし、実際のアプリケーションで n はとても大きく、$\lceil \frac{W_{max}}{W_{min}^2} \rceil$ よりも大きな値となる。そのため、LSiX の必要な時間は L-BiX よりも小さくなる。他にも L-BiX はクエリごとに配列を持つため、LSiX よりも多くのス

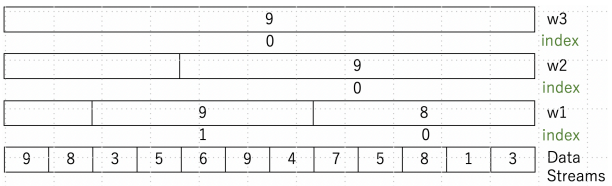


図 2: window サイズが $w1 = 5, w2 = 8, w3 = 12$ で slide サイズが 1 の 3 つのクエリを求める MCQA の例

表 1: 必要な時間とスペース

Algorithm	Time	Space
L-BiX	$3n$	$\sum_{i=1}^{n}(\frac{W_i}{Sl})$
MCQA	$\sum_{i=2}^{n}(\lfloor\frac{W_i}{W_{i-1}}\rfloor + W_i\%W_{i-1}) + \lfloor\frac{W_1}{Sl}\rfloor + W_1\%Sl$	$\lceil\frac{W_{max}}{Sl}\rceil \times n$
LSiX	$2n+3+3\lceil\frac{W_{max}}{W_{min}^2}\rceil$	$W_{max} + \lceil\frac{W_{max}}{W_{min}}\rceil$

ペースを必要とする。MCQA は一つずつ window を check する必要があり、必要な時間は連続する 2 つの window に強く影響され、非効率である。また、必要なスペースもクエリ数に対して過大である。これに対して LSiX は W_{max} をチェックするだけでよく、全てのクエリで他の window を無視できるため、必要な時間は MCQA よりも大幅に短い。同様に LSiX の必要なスペースは MCQA よりも n 倍近く小さい。

4. 提案手法

この章では *Longest-Shortest-window-based Indexing(LSiX)* という複数クエリの SWAG を効率的に処理する手法を提案する。

4.1 LSIX の概要

ユーザーに定義された SWAG が与えられた時、最長の window サイズを W_{max}, 最短の window サイズを W_{min} とする。この時、LSiX はサイズが W_{max} となる Lo と $\lceil\frac{W_{max}}{W_{min}}\rceil$ となる Sh の二つの配列のみを使用して部分集約を保持する。

- “Lo” は $W_{min}(< W_{max})$ 以下で全ての部分集約を保持する。Lo のインデックスが W_{min} を超えるとサイズ W_{min} の新しいサイクルが始まり、現在のサイクルの部分集約が更新されていく。あるクエリの結果を計算するときに、Lo の値はそのクエリの両端の値として使用される。
- “Sh” は Lo における各サイクルの最大値を集約した値を保持する。あるクエリの結果を計算するときに、Sh の値はそのクエリの中央の値として使用される。

LSiX では、Lo と Sh は新しいレコードが来るたびに更新される。そして、各クエリの結果は図 3 のように Lo から両端のために二つの値と Sh から中央のための値を集約することによって求めることができる。これは LSiX が window サイズに関わらず、最大 2 回の集約操作でクエリの結果を求めることができることを意味する。これによって、クエリの結果を計算するのに必要な集約操作の合計回数は MCQA よりも大幅に少なくなる。

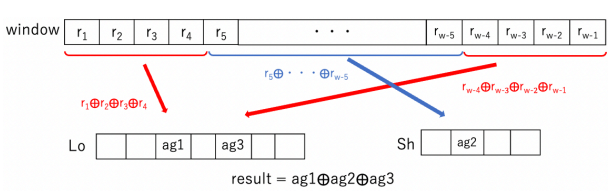


図 3: Lo と Sh を使って、window サイズ:w のクエリを求めるイメージ図。($\oplus$ は集約関数を表す)

簡略化のため、以下の議論では全てのクエリが slide サイズ:1 であるとする。しかし、提案手法では上記とは異なるクエリにも対応しており、これは Section 4.7 で説明する。加えて、LSiX は count-based な window と time-based な window の両方に対応している。

4.2 データ構造

LSiX は二つの索引付き配列を使用し、全てのクエリを計算するための部分集約を効率的に保持する。LSiX の構造は Def. 1 のように定義する。

Definition 1. *n 個のクエリと集約関数 $\oplus$, 最長の window:W_{max}, 最短の window:W_{min} が与えられると LSiX の構造は以下の二つの配列で構成される：*

- *“Lo” of size $b = W_{max}$*
- *“Sh” of size $c = \lceil\frac{W_{max}}{W_{min}}\rceil$*

window サイズが $W_{min} = w1 = 5$, $w2 = 8$, $W_{max} = w3 = 12$ で max を求める 3 つのクエリを集約する LSiX の構造を例とする。この時、Lo のサイズは 12、Sh は $\lceil\frac{12}{5}\rceil = 3$ となる。

4.3 *Lo* の集約

まず、Lo の集約を説明する前に重要な概念を説明する。

変数 p p は Lo の index を表す変数。p は新しいレコードが到着するたびに 1 ずつ増えていき、最後に 0 に戻る。($Lo[W_{max}-1]$ など)

サイクル Lo は複数のサイクルで構成され、各サイクルは Def. 2 で定義されたサイズ:W_{min} の部分配列である。

Definition 2. *$Lo[0]$ から順に W_{min} ずつサイクルに分けられる（最後のサイクルのみ M_{min} ではない可能性がある）。Lo に含まれる各要素はサイクル内での位置付けによって、以下の 3 つに分類される。*

- *st:* サイクルの始まりとなる要素

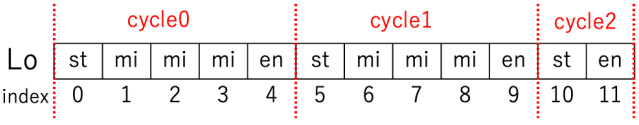


図 4: window サイズが $w1 = 5, w2 = 8, w3 = 12$ の 3 つのクエリを求める Lo の例

- *en:* サイクルの終わりとなる要素
- *mi: st,en* 以外の要素

図 4 は window サイズが $W_{min} = w1 = 5$, $w2 = 8$, $W_{max} = w3 = 12$ の 3 つのクエリを集約する Lo の例である。この例では $W_{min} = 5$ のため、$Lo[0] \sim Lo[4]$, $Lo[5] \sim Lo[9]$, $Lo[10] \sim Lo[11]$ の 3 つのサイクルで構成される。また、Lo は p の位置によって以下のように集約される。ただし、R_{new} は最新のレコード, R_{pre} は一つ前のレコードを指す。

(Rule 1) $Lo[p]$ が st の時、

$Lo[p] = R_{new}$.

(Rule 2) $Lo[p]$ が mi の時、

$Lo[p] = R_{new} \oplus Lo[p-1]$;

$Lo[p-1] = R_{pre}$.

(Rule 3) $Lo[p]$ が en の時、

$Lo[p] = R_{new}$;

$Lo[p-1] \oplus = Lo[p]$;

$Lo[p-2] \oplus = Lo[p-1]$;

$\vdots$

$Lo[p - W_{min} + 1] \oplus = Lo[p - W_{min} + 2]$.

Lo は新しいレコードが到着し、p の値が更新されるたびに上記のような集約を行うことで、部分集約を保持していく。

4.4 *Sh* の集約

Sh は各サイクルの st の値を集約した値が格納される配列である。Sh の index は Lo のサイクルと左から順に 1 対 1 で対応している。例として図 4 を用いると、cycle0 は $Sh[0]$ に、cycle1 は $Sh[1]$ に、cycle2 は $Sh[2]$ に対応している。Sh では p が en に到達したときにのみ、以下のような集約が実行される。(式中の k は、k=$\lfloor p/W_{min} \rfloor$ で、p のあるサイクルと対応した Sh の index を示す)

(Rule 4) $Sh[k] = Lo[k * W_{min}]$

$Sh[(k-1)/c] = Sh[k] \oplus Lo[((k-1)/c) * W_{min}]$

$\vdots$

$Sh[(k-c+1)/c] = Sh[(k-c+2)/c] \oplus Lo[((k-c+1)/c) * W_{min}]$

Rule4 は p のあるサイクルと対応した $Sh[k]$ から順に Sh 全体を更新する。結果として、$Sh[(k-c+1)/c]$ には、W_{max} のクエリを集約した値が格納される。Lo と Sh の集約の詳細は Algorithm 1 のようになる。

LSiX では、Lo と Sh から適切な値を使用することで最大 2 回の集約操作で各クエリの結果を求めることができる。

4.5 集約の例

Lo と Sh で集約を行う例として、window サイズが $W_{min} = w1 = 5$, $w2 = 8$, $W_{max} = w3 = 12$ で max を求める 3 つのクエリを処理する。この例は図 5 で示す。

step 0 (p=11) である初期状態では、Lo の各サイクルで Rule 3 が行われ、Sh では Rule4 が行われた結果が格納されている。step1 では新しいレコード (3) が到着し、p=0 となる。$Lo[0]$ は st に分類されるため Rule1 が実行される

Algorithm 1 Aggregation

INPUT: n queries with $\oplus$ aggregation and slide size Sl, W: all windows, W_{max}: longest window, W_{min}: shortest window, $b = \lceil \frac{W_{max}}{Sl} \rceil$, $s = \lceil \frac{W_{min}}{Sl} \rceil$, Lo and Sh created by Def. 1, and data stream DS

OUTPUT: result

```
1: p=0, A^p_slide=""
2: for A_slide from DS do
3:     if p%s== s-1 or p == b-1 then
4:         agg = A_slide ⊕ A^p_slide
5:         Lo[p] = A_slide, i = 0
6:         for i = p-1; i%s!=0 do
7:             Lo[i] = agg
8:             agg = agg ⊕ Lo[i-1]
9:             i–
10:        end for
11:        l = ⌊ p/s ⌋
12:        Lo[i] = agg, Sh[l] = agg
13:        if W_max%W_min == 0 then
14:            len = ⌊ W_max/W_min ⌋
15:        else
16:            len = ⌊ W_max/W_min ⌋ + 1
17:        end if
18:        while j = 0; j <len-1 do
19:            l = (l+len-1)%len
20:            agg = agg ⊕ Lo[l*s]
21:            Sh[l] = agg
22:            j++
23:        end while
24:    else if p%s = 0 then
25:        Lo[p] = A_slide
26:    else
27:        Lo[p] = Lo[p-1] ⊕ A_slide
28:        Lo[p-1] = A^p_slide
29:    end if
30:    A^p_slide = A_slide
31:    result = CREATE_RESULT(p, Sl, W, b, s, Lo, Sh)
32:    p++
33:    p=0 if p==b
34: end for
```

($Lo[0]$=3)。次に step2 では新しいレコード (1) が到着し、p=1 となる。$Lo[1]$ は mi に分類されるため Rule2 が実行される ($Lo[1] = max(Lo[0], 1) = 3, Lo[0] = 3$)。step3(p=2) と 4(p=3) の時は、mi に分類されるため、Rule 2 が行われた結果が格納される。step5 では新しいレコード (4) が到着し、p=4 となる。$Lo[4]$ は en に分類されるため Rule3 と Rule4 が実行される。

Rule3 では $Lo[4]$ の所属する赤色のサイクルが全て更新され、

$Lo[4] = 4$,

$Lo[3] = max(Lo[4], Lo[3]) = max(4, 6) = 6$,

$Lo[2] = max(Lo[3], Lo[2]) = max(6, 5) = 6$,

$Lo[1] = max(Lo[2], Lo[1]) = max(6, 1) = 6$,

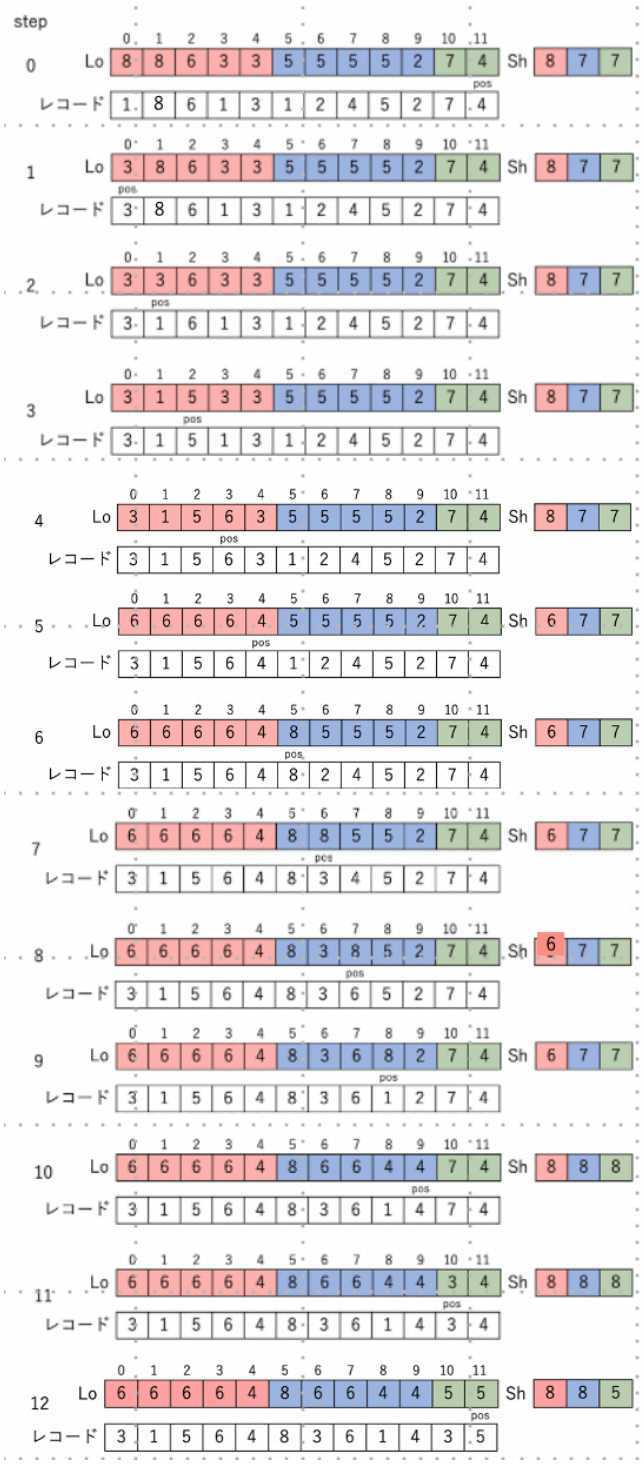


図 5: w1=5,w2=8,w3=12 のクエリを集約する Lo と Sh の例.

$Lo[0] = max(Lo[1], Lo[0]) = max(6, 3) = 6$ となる。その後、赤色のサイクルと対応する $Sh[\lfloor 4/5 \rfloor] = Sh[0]$ から Rule4 が実行され、$Sh[0] = Lo[0] = 6$,
$Sh[2] = max(Sh[0], Lo[10]) = max(6, 7) = 7$,
$Sh[3] = max(Sh[2], Lo[5]) = max(7, 5) = 7$ となる。これ以降でも、新しいレコードが到着したら p が 1 追加され、適切な処理が実行されるというのを繰り返す。

LSiX では、Lo と Sh の部分集約を使用することで window サイズが $W_{min} \sim W_{max}$ のクエリの結果を求めることができる。例えば step1 の時、Q[12,1] の結果は $max(Lo[0], max(Sh[1], Lo[1])) = 8$ と 2 回の集約操作で求めることができる。また、step9 の時、Q[5,1] の結果は $Lo[5] = 8$, Q[8,1] は $max(Lo[2], Sh[1]) = 8$ と 2 回以下の集約操作で求めることができる。このように LSiX では、Lo と Sh の最適な値を使うことで、各クエリの結果を 2 回以下の集約操作で求めることができる。クエリ処理の詳細は Algorithm 2 のようになる。

Algorithm 2 Function that return the results of each query

```
1: function CREATE_RESULT(n, p, Sl, W, b, s, Lo, Sh)
2:     if ⌊ qe/s ⌋ b%s = 0 then
3:         D = b
4:     else
5:         D = (⌊b/s⌋ + 1) * s
6:     end if
7:     for i = 0 ∼ n do
8:         qe = (p + b + 1 − ⌈W[i]/Sl⌉)%b
9:         x = ⌊ [(qe+s)%D]/s ⌋
10:        y = ⌊ (p%D)/s ⌋
11:        if (p%s = s-1 or p = b-1) and qe%s = 0 then
12:            result[i] = Sh[⌊ qe/s ⌋]
13:        else if p%s = s-1 or p = b-1 then
14:            result[i] = Sh[x] ⊕ Lo[qe]
15:        else if qe%s = 0 then
16:            result[i] = Sh[⌊ qe/s ⌋] ⊕ Lo[p]
17:        else if x = y then
18:            result[i] = Lo[qe] ⊕ Lo[p]
19:        else
20:            result[i] = Lo[p] ⊕ Lo[qe] ⊕ Sh[x]
21:        end if
22:    end for
23:    return result
24: end function
```

4.6 必要な計算量とメモリ

LSiX に必要な時間とスペースについて詳しく説明する。概要はテーブル 1 の通りである。

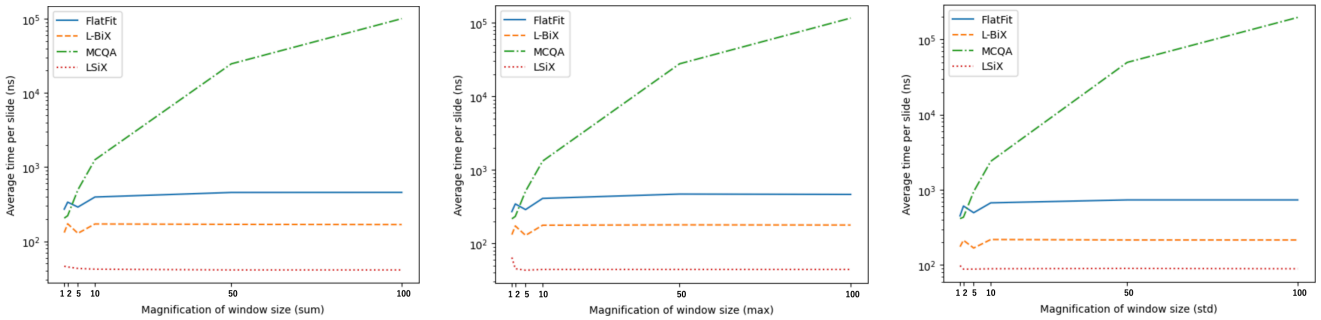
最長と最短の window サイズが W_{max} と W_{min} の n 個のクエリが与えられた時、LSiX のコストは以下のようになる。

- Time: $2n + 3 + 3\lceil \frac{W_{max}}{W^2_{min}} \rceil$
- Space: $W_{max} + \lceil \frac{W_{max}}{W_{min}} \rceil$.

Proof. W_{max} 回の間に Lo と Sh の更新と全てのクエリを計算するのに必要な集約操作は以下のように求まる。

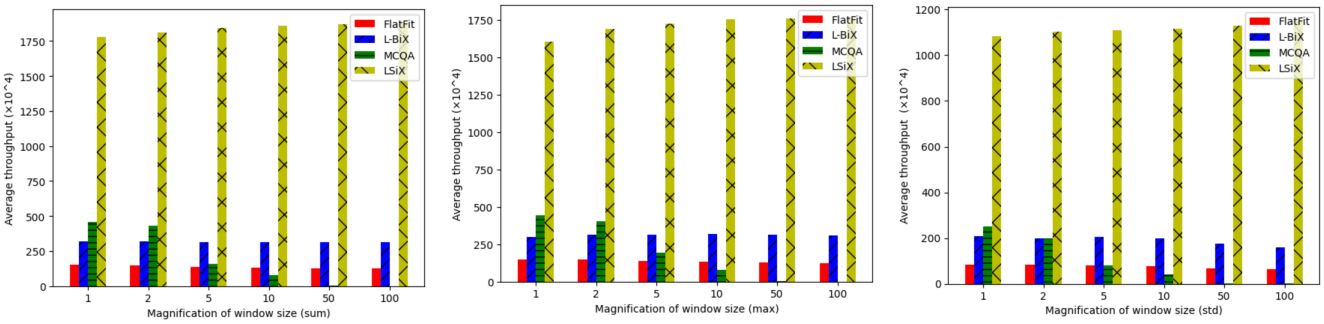
- 各クエリの結果を計算するには最大で 2 回の集約操作が必要となる。そのため、W_{max} 回の間に n 個のクエリに必要な集約操作の合計回数は、$2nW_{max}$ となる
- Lo での後方集約の回数は、$2W_{max}$
- Lo での前方集約の回数は、W_{max}
- Sh での後方集約の回数は、$3\lceil \frac{W_{max}}{W_{min}} \rceil^2$

それゆえ、集約操作の合計は、$2nW_{max} + 3W_{max} + 3\lceil \frac{W_{max}}{W_{min}} \rceil^2$。これを W_{max} で割り平均を求めると、



(a) sum (b) max (c) std

図 6: count-based window に対する window サイズを変化させた実行時間の結果



(a) sum (b) max (c) std

図 7: count-based window に対する window サイズを変化させた処理能力の結果

$\frac{2nW_{max}+3W_{max}+3\lceil\frac{W_{max}}{W_{min}}\rceil^2}{W_{max}} = 2n+3+3\lceil\frac{W_{max}}{W_{min}^2}\rceil$ となり、計算量が求まる。

LSiX はサイズが W_{max} の Lo とサイズが $\lceil\frac{W_{max}}{W_{min}}\rceil$ の Sh の二つの配列のみを必要とする。そのため、$W_{max}+\lceil\frac{W_{max}}{W_{min}}\rceil$ とスペースが求まる。

□

5.7 異なる slide サイズを持つ windows

もし、slide サイズで割り切れない window サイズを持つクエリや異なる slide サイズのクエリがある時、slide サイズを 2 つ以上に分割し、同じ部分のレコードが同時に期限切れになるようにする必要がある。これにより、期限切れのレコードを効率的にパージすることができる。

提案する方法は全ての window と slide の最大公約数 ($gcd = GCD(Windows, Slides)$) を利用することだ。そして、$gcd$ は W_{max} と W_{min} を分割するために使用される。Lo のサイズは $b = \lceil\frac{W_{max}}{gcd}\rceil$、$Sh$ のサイズは $c = \lceil\frac{W_{max}}{W_{min}}\rceil$ となる。Lo と Sh を更新する回数は、レコードが到着するたびにではなく、gcd 個のレコードを集約した値が来るたびに更新されるため、減らされる。そして、Algorithm 1 と同じ手順でデータストリームに対するクエリ処理が行われる。

5. 実　験

5.1 実験環境

提案手法の有効性を比較するために 1slide あたりの平均処理時間で比較した。比較には、複数クエリを処理する最先端の手法である MCQA と単一のクエリを処理する最先端の手法である L-BiX と FlatFit を用いた。

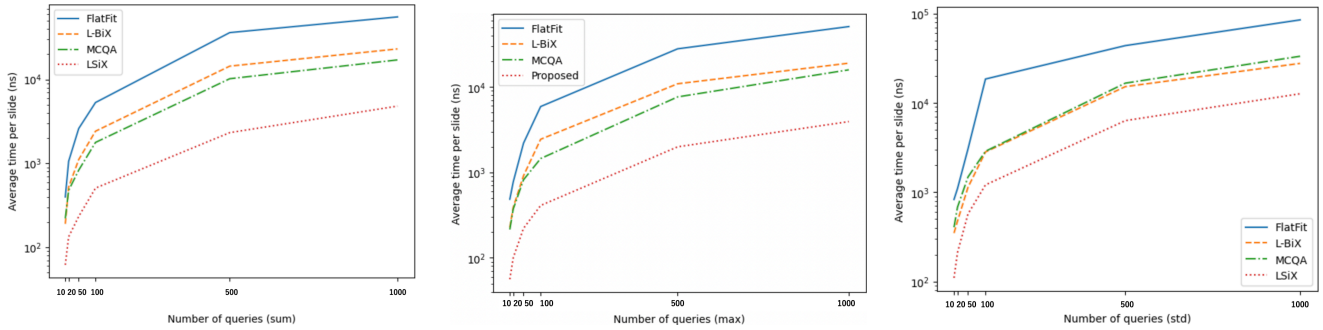
検証には、DEBS データセットを使用した。DEBS12Grand Challenge Datatset (DEBS) は同様の研究フレームワークでクエリ処理負荷評価に広く使用されている標準データセットである。このデータセットは工場内のさまざまな大規模センサーによって生成された記録から構成されている。レコードは 51 次元となっていて、実験にはそのうちに 1 次元を採用した。データセットは約 3200 万レコードを含む。

クエリ処理の負荷は SWAG のパフォーマンスに影響を与える。それゆえ、window サイズを変化させた実験とクエリの数を変化させた実験を行った。LSiX は time-based window と count-based window を共に処理できるが、実験では count-based window のみを使用した。実験の結果はそれぞれのアルゴリズムの 5 回の独立した実行の平均とする。

集約関数としては、Distributive 関数である sum, max と algebraic 関数である標準偏差 (std) を使用した。すべての実験は Apple M1 Pro、16GB のマシンで行った。

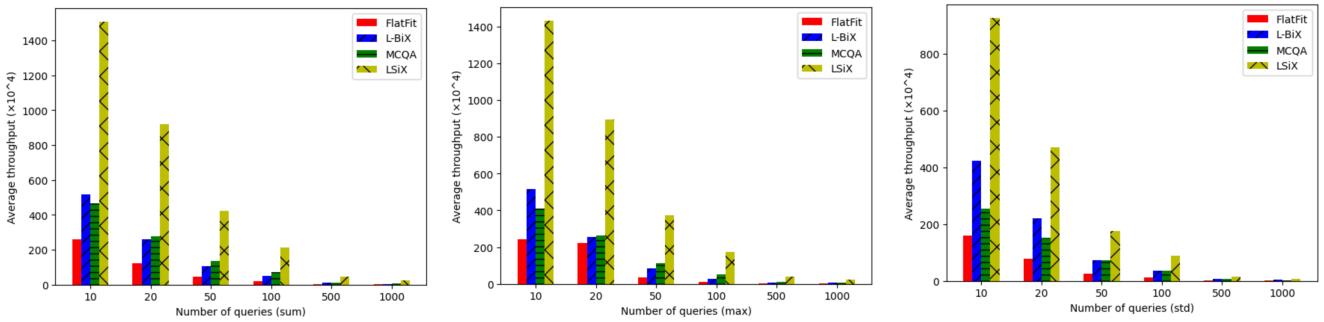
5.2　window サイズを変化させた実験

この実験では slide サイズを 10 に固定して、window サイズを変化させて実験を行った。window サイズが W=[35, 65, 75, 100, 110, 160, 205, 220, 235, 280] となる 10 個のクエリを 1,2,5,10,50,100 でスケールさせた。実験の結果は図 6, 7 に示す。



(a) sum (b) max (c) std

図 8: count-based window に対するクエリの数を変化させた実行時間の結果



(a) sum (b) max (c) std

図 9: count-based window に対するクエリの数を変化させた実行時間の結果

一般的に、処理能力は window サイズが増加するにつれて減少する傾向にある。加えて第 3 章で説明したように、window サイズと slide サイズの比が大きくなったり、連続する二つの window サイズが大きくなるにつれて MCQA の処理能力は大幅に減少する。また、window サイズと共に後方集約の回数が増えるため、L-BiX と FlatFIT の処理能力も軽く減少する。それに対して、LSiX の処理能力は window サイズが増加するにつれて増加している。これは、window サイズが大きくなると Sh を更新する回数である $\lceil \frac{W_{max}}{W^2_{min}} \rceil$ が減少するためである。これにより、LSiX と他の手法の処理時間の差は window サイズが増えるにつれて顕著になる。

具体的に比べると、window サイズを 1 倍しているときに、LSiX は MCQA よりは 4 約倍速く、L-BiX よりは約 5 倍速く、FlatFIT よりは約 9 倍速い。次に window サイズを 100 倍しているときに、LSiX は MCQA よりは 1000 約倍速く、L-BiX よりは約 6 倍速く、FlatFIT よりは約 14 倍速い。

5.3 クエリの数を変化させた実験

この実験では slide サイズを 10 に固定して、クエリの数を n=10,20,50,100,500,1000 と変化させた。window サイズは次のように決められる。

$$w_1 = 10$$
$$w_i = w_{i-1} + a \qquad l$$
(a は $[10, 15, 20, 25, 30, 35, 40, 45, 50]$ の中のランダムな値)

実験の結果は図 8, 9 に示す。

全ての手法でクエリの数が増加するにつれて処理時間は増加し、処理能力は減少している。しかし、LSiX はクエリの数に依存する集約操作と配列操作の回数が他の手法よりも少ないため、最良の結果となっている。

具体的に比べると、n=10 のときに、LSiX は MCQA よりは 3.5 約倍速く、L-BiX よりは約 3 倍速く、FlatFIT よりは約 8 倍速い。次に n=1000 のときに、LSiX は MCQA よりは 4 約倍速く、L-BiX よりは約 5 倍速く、FlatFIT よりは約 11 倍速い

6. 結　論

本稿では、単一のデータストリームに対して、複数クエリを効率的に処理するための手法を提案した。提案手法は最長と最短の window サイズをもとにした二つの配列 Lo,Sh で集約を行い、その結果を全てのクエリで共有することで効率的に求める。配列 Lo,Sh は前方集約と後方集約の二つを組み合わせることで、最小限の集約操作の回数で範囲内の部分集約を保持する。結果として、クエリの数に依存するワークロードを少なくし、全てのクエリを高速に求めることができる。

実験の結果、提案手法が最新の手法である MCQA と L-BiX, FlatFIT にスループットで大幅に上回ることがわかった。提案手法は少なくても MCQA より 3.5 倍早く、L-BiX よりすくな 3 倍早く、FlatFit より 8 倍早く処理する。

将来の展望としては、(1)LSiX の計算量が W_max に依存しないように拡張する、(2) 順不同のデータストリームに対して、複数クエリを処理できるように拡張する、の二つがある。

7. 謝　辞

この成果は，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の 「ポスト 5G 情報通信システム基盤強化研究開発事業」（JPNP20017）の委託事業，JST CREST（JPMJCR22M2），科学研究費補助金（JP22H03694, JP23K16888）に支援によるものです.

文　献

[1] Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB J.*, 12(2):120–139, 2003.

[2] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2):121–142, 2006.

[3] Savong Bou, Toshiyuki Amagasa, and Hiroyuki Kitagawa. Intrans: Fast incremental transformer for time series data prediction. In Christine Strauss, Alfredo Cuzzocrea, Gabriele Kotsis, A Min Tjoa, and Ismail Khalil, editors, *Database and Expert Systems Applications - 33rd International Conference, DEXA 2022, Vienna, Austria, August 22-24, 2022, Proceedings, Part II*, volume 13427 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2022.

[4] Savong Bou, Hiroyuki Kitagawa, and Toshiyuki Amagasa. L-bix: incremental sliding-window aggregation over data streams using linear bidirectional aggregating indexes. *Knowl. Inf. Syst.*, 62(8):3107–3131, 2020.

[5] Paris Carbone, Jonas Traub, Asterios Katsifodimos, Seif Haridi, and Volker Markl. Cutty: Aggregate sharing for user-defined windows. In Snehasis Mukhopadhyay, ChengXiang Zhai, Elisa Bertino, Fabio Crestani, Javed Mostafa, Jie Tang, Luo Si, Xiaofang Zhou, Yi Chang, Yunyao Li, and Parikshit Sondhi, editors, *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 1201–1210. ACM, 2016.

[6] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997.

[7] Sailesh Krishnamurthy, Chung Wu, and Michael J. Franklin. On-the-fly sharing for streamed aggregation. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 623–634. ACM, 2006.

[8] Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. *SIGMOD Rec.*, 34(1):39–44, 2005.

[9] Wen Liu, Tuqian Zhang, and Junxia Liu. Window-based multiple continuous query algorithm for data streams. *J. Supercomput.*, 75(9):5782–5807, 2019.

[10] Yu Ma, Rui Mao, Qika Lin, Peng Wu, and Erik Cambria. Multi-source aggregated classification for stock price movement prediction. *Inf. Fusion*, 91:515–528, 2023.

[11] Anatoli U. Shein, Panos K. Chrysanthis, and Alexandros Labrinidis. Flatfit: Accelerated incremental sliding-window aggregation for real-time analytics. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Chicago, IL, USA, June 27-29, 2017*, pages 5:1–5:12. ACM, 2017.

[12] Kanat Tangwongsan, Martin Hirzel, and Scott Schneider. Low-latency sliding-window aggregation in worst-case constant time. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*, pages 66–77. ACM, 2017.

大規模データストリームに対する高速なS-FINCHクラスタリング

牛尼　索造†　　藤原　靖宏††　　塩川　浩昭†††

† 筑波大学 情報学群 情報科学類　〒 305–8577 茨城県つくば市天王台 1-1-1
†† NTT コミュニケーション科学基礎研究所　〒 243-0198 神奈川県厚木市森の里若宮 3-1
††† 筑波大学 計算科学研究センター　〒 305–8577 茨城県つくば市天王台 1-1-1
E-mail: †sushiama@kde.cs.tsukuba.ac.jp, ††yasuhiro.fujiwara@ntt.com, †††shiokawa@cs.tsukuba.ac.jp

あらまし　クラスタリングはデータ集合に内在する性質の類似した部分集合を検出する手法であり，データ分析において重要な要素技術となっている．その中でも近年注目を集めているのは，データストリームに対して凝集型階層的クラスタリングを行う S-FINCH である．S-FINCH は時々刻々と到来するデータストリームに対して各データオブジェクトの最近傍や共有最近傍を求める必要があるため，各データオブジェクト間の距離計算が必要となる．そのため，S-FINCH はクラスタリングにおいて膨大な計算時間を必要とする．そこで本稿では S-FINCH の高速化手法を提案する．提案手法は空間索引を利用することで，S-FINCH において生じる距離計算回数を削減する．本稿では実データを用いた評価実験により，提案手法は S-FINCH に対してクラスタリング精度を損なわずに効率的なクラスタリング処理が実行可能であることを示した．

キーワード　ストリームデータ処理，空間・時空間・時系列データ処理，データ構造・索引

1　はじめに

多次元データストリームに対するクラスタリングとは膨大なデータから性質の類似するデータごとのグループ分割を提供する手法である．多くの科学の分野において，膨大なデータから性質の類似するデータのグループを見つけだしラベル付けをすることは非常に重要になっている．特に近年では，取り扱われるデータのサンプル数や次元数がますます増加し，このようなデータに対する高速な解析処理技術への需要が高まっている．膨大なデータから性質の類似するデータのグループ分割を提供する手法のひとつとして，クラスタリングが挙げられる．

近年 Sarfraz らによって FINCH [1] というクラスタリング手法が提案された．FINCH は多次元のデータポイントの集合を入力として受け取り，その各データポイントの間の距離を計算し，最も近くにあるデータポイントとの間，または同じデータポイントを最も近いとするデータポイント間にエッジを張ることで得られるグラフの各連結成分を 1 つのグループとする．この各グループ内の各データポイントの平均を代表点とし，次の入力として同じ処理を繰り返す．これにより，異なる粒度のグループ分けを持つクラスタ階層を出力する．FINCH は従来の k-means 法 [2] などのクラスタリング手法と比較して，1.) 高い精度で真のクラスタを自動的に決定できる，2.) ハイパーパラメータを必要としない，3.) 異なるドメインのデータでも一般的に利用できる，4.) 膨大なデータセットに拡張できるなどのメリットがあり，高次元のデータセットに対してもハイパーパラメータを必要とせずに高速なクラスタリングが実現可能であることから，FINCH は画像や化学などの幅広い分野での応用 [3–5] に挑戦されている．

しかし，FINCH は静的なデータを対象としたアルゴリズムであるため，データストリームを処理しようとした場合，データが追加されるとデータ全体に対して最近傍探索を行いクラスタ構造を再構築する必要がある．そのため，FINCH はデータストリームのクラスタリングに膨大な処理時間を必要とする．この問題に対処するために Cunningham によって S-FINCH [6] が提案された．S-FINCH は，データが追加された時にクラスタ構造を全て再構築するのではなく，追加されたデータによって変更が生じた部分のみを対象として再計算することによって処理時間を短縮する．

しかしながら，S-FINCH は FINCH に比べて高速なストリーム処理が可能ではあるものの，大規模なストリームデータを対象としたクラスリングは困難である．S-FINCH では，追加されたデータと全ての頂点間に対して距離を計算する．そのため，n 件目のデータが追加された時に，距離計算のみで $\mathcal{O}(n)$ の時間計算量が生じる．また，S-FINCH は階層的クラスタリング手法であるため，データの追加によって他の階層のクラスタ構造も変化する．この階層は高々 $\log_2 n$ であることから，この処理は $\mathcal{O}(n \log n)$ の時間計算量を要する．N 件のストリームデータを処理するのに $\mathcal{O}\left(N^2 \log N\right)$ を要する．したがって，大規模なストリームデータに対しては膨大な処理時間を要することになる．

1.1　本研究の貢献

本稿では S-FINCH の高速化手法を提案する．提案手法では従来手法 S-FINCH の計算コストを削減するために，FINCH が提供するクラスタ構造が最近傍連結成分という小規模な集合によって構成されていることに着目する．また，FINCH で生成されたクラスタは他の最近傍クラスタを持ち最近傍関係にある連結成分が上の階層でのクラスタとなるため，各クラスタの関係を木構造として表現出来る．最近傍関係にある小規模な集

合を，特別に構築することなく得られるため，この集合による空間を空間索引として活用することで，空間索引の構築コストを削減できると考えられる．

そこで本稿では各クラスタに対し各データを包含する空間を求め，最近傍や逆最近傍を求めたいデータポイントとその空間との最短距離を各クラスタの各データの点との距離の下限とし，その下限を利用して最近傍・逆最近傍にならないデータポイントを特定することで距離計算回数を削減する．その結果として提案手法は以下の特性を示す．

計算回数

S-FINCH に比べ少ない計算回数で最近傍および被最近傍探索を行うことができる．

正確性

提案手法で用いるアプローチは，厳密な最近傍を求めるため FINCH が提供するクラスタ構造から精度を損わない．

本稿の構成は，次の通りである．2 節で本稿の前提となる知識について概説する．3 節にて提案手法の詳細について説明し，4 節において提案手法の評価と分析を行う．5 節にて，本稿をまとめ，今後の課題について論ずる．

2 事 前 準 備

この節では FINCH [1] とストリーム処理を可能にした S-FINCH [6] について説明する．表 1 に主な記号とその定義を示す．

FINCH [7] は，d 次元のデータポイントが N 件ある多次元データセット $\boldsymbol{S} \in \mathbb{R}^{N\times d}$ を入力として受け取る．これに対して，S-FINCH は入力として d 次元のデータポイントが N 件連続している多次元データストリームを $\boldsymbol{L} = [L_1, L_2, \ldots, L_N]$（ただし，$L_i \in \mathbb{R}^d$）を受け取る．距離尺度として，FINCH および S-FINCH はユークリッド距離を用いる．すなわち，2 つのデータポイントのユークリッド距離が小さいとき，それらは類似したデータポイントであることを意味する．多次元データポイント v, w があるとき，これらの距離を $dist(v, w)$ を表記する．加えて，多次元空間 X があるとき，v から X までの距離の最小値を $min\_dist(v, X)$ を表記する．データポイント x の最近傍とは，x に最も近いデータポイント y のことを指し $NN(x) = y$ と表記するこれに対して，データポイント x の被最近傍とは，x が最近傍であるデータポイント集合 $NNset(x)$ のことを指し $NNset(x) = \{y \mid NN(y) = x\}$ 定義する．

FINCH [7] および FINCH はクラスタリング結果として階層的なクラスタ構造を出力する．そしてこのクラスタ構造の階層の数，つまり階層の高さを H とする．本論文ではこのクラスタ構造の一番下の階層を高さ 1 と，一番上の階層を高さ H とする．それぞれの手法に入力として与えられたデータの各データポイントはまず高さ 1 の階層でクラスタを構成することになる．ここで入力として与えられた各データポイントは 1 つのデータポイントからなるクラスタとして扱う．これにより，高さ 1 の階層ではクラスタ構造の全てのデータポイントは全てクラスタの代表点となる．最上層ではない階層においてデータポイント x がクラスタ X_3 に属するとき，クラスタ X_3 のことを x の親と呼び，$parent(x) = X_3$ と表記する．また，x はクラスタ X_3 の子となり，$x \in children(X_3)$ と表記する．データポイント x の子となるデータポイント $children(x)$ や，さらに $children(x)$ の子となるデータポイント $children(children(x))$ 全てを子孫データポイントと呼ぶ．つまり，データポイント x の子孫となるデータポイントの集合を $descendants(x)$ とするとき，$children(x) \in descendants(descendants)$ であり，さらに $descendants(children(x)) \in descendants(x)$ となる．データポイント x の親となるデータポイント $children(x)$ や，さらに $children(x)$ の親となるデータポイント $children(children(x))$ 全てを祖先データポイントと呼ぶ．つまり，データポイント x の祖先となるデータポイントの集合を $ancestor(x)$ とするとき，$children(x) \in ancestor(ancestor)$ であり，さらに $ancestor(children(x)) \in ancestor(x)$ となる．

表 1: 記号と定義

記号	定義
N	データポイントの数
n	あるステップに与えられたデータポイントの数
d	データポイント x の次元数
κ_i^1	データポイント i の 最近傍データポイント
$L(x)$	データポイント x のクラスタラベル
$parent(x)$	高さ i のデータポイント x のクラスタの 高さ $i+1$ での代表データポイント
$children(x)$	$\{y \mid parent(y) = x\}$
$dist(v, w)$	データポイント v, w の距離
$min\_dist(v, W)$	データポイント v と空間 w の距離の最小値
$NN(x)$	x の最近傍データポイント
$NNset(x)$	x が最近傍であるデータポイント
$V_{h,sh,x}$	高さ h のデータポイント x の子孫のうち， 高さ sh にあるデータポイントを包含する空間
$W_{h,x}$	高さ h のデータポイント x を中心とし， x の最近傍までの距離を半径とする超球
$S_{h,sh,x}$	高さ h のデータポイント x の子孫のうち， 高さ sh にあるデータポイント y の $W_{h,y}$ を包含する空間

2.1 FINCH

FINCH は， Sarfraz らによって提案された出力するクラスタ数やハイパーパラメータの指定を必要とせず，データポイントの近傍関係を利用して凝集型階層的クラスタリングを行う手法である．FINCH は与えられたデータセットからグラフを生成し，各連結成分を 1 つのクラスタとすることを繰り返しクラスタ階層を生成する．生成されたクラスタの数が 1 ではないとき，クラスタの構成要素となる各データポイントの平均，つまり重心を代表点とする．この各クラスタの代表点の集合を次の処理でのデータセットとする．与えられたデータセットをクラスタリングすることを 1 つのステップとし，ステップを繰り返すことで階層的なクラスタ構造を生成する．

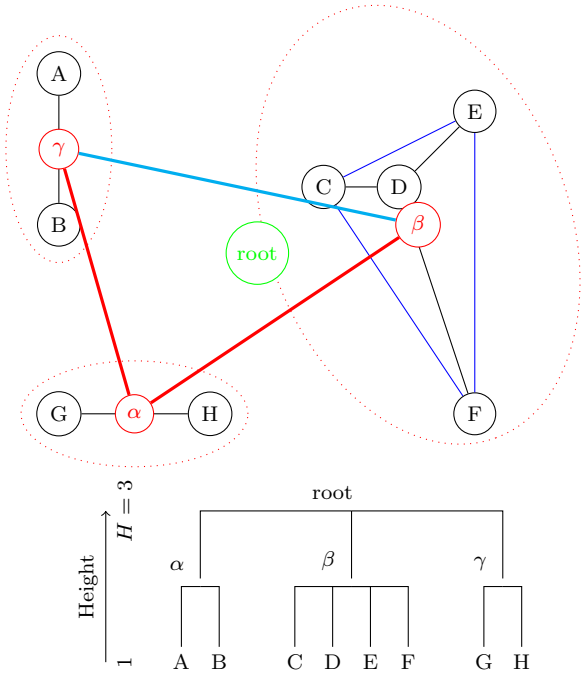


図 1: FINCH でクラスタを生成するときのグラフと対応するデンドログラム

次に連結成分を求めるために生成するグラフについて説明する．データセットの各データポイント間の距離を計算し，最近傍を求め，式 (1) で求まる隣接行列からグラフを生成する．

$$A(i,j) = \begin{cases} 1 \ if \ j = NN(i) \quad or \ j = NN(j) \\ \qquad\qquad\qquad\quad or \ NN(i) = NN(j) \\ 0 \ Otherwise \end{cases} \tag{1}$$

入力例とグラフの例を図 1 に示す．A から H は FINCH に入力として与えられる各データポイント，α から β は次のステップにおいて入力として与えられる各クラスタの構成要素の平均データポイントである．C, E, F を互いに結ぶ青い線，及び α, β を結ぶ水色の線は最近傍関係にはないものの同じデータポイントを最近傍に持つデータポイントの間に張られるエッジを示している．これは式 (1) において $NN(i) = NN(j)$ となる場合に該当する．

FINCH の時間計算量を解析するために，各ステップでの時間計算量を求める．あるステップの入力として n 件のデータポイントがある場合，最近傍探索に $\mathcal{O}(n^2 d)$，連結成分の探索に $\mathcal{O}(n)$ の時間計算量が必要となり，1 ステップに $\mathcal{O}(n^2 d)$ の時間計算量を要する．各データポイントはそれぞれの最近傍と同じの連結成分に含まれ 1 つのクラスタとなるため，ステップごとに入力となるデータセットのサイズは半分以下になる．そのため，FINCH に与えられた N 件の入力データセットに対して階層の数は高々 $\lceil \log_2 N \rceil$ となり，全体としての時間計算量は $\mathcal{O}(N^2 d \log N)$ となる．

2.2 S-FINCH

FINCH における逐次的なデータ更新を行うストリーム処理を達成することを目的に，近年 Cunningham によって S-FINCH が提案された．FINCH はデータの追加に対応できるクラスタリング手法ではないため FINCH を愚直にストリーム処理に対応させると，データが追加される度に過去のデータも含め全体のデータセットに対して FINCH を実行する必要があり $\mathcal{O}(N^3 d \log N)$ の時間計算量を要する．そこで S-FINCH では，データが追加される前のクラスタ構造を用いることで時間計算量を大きく改善する．

S-FINCH は 3 つのステージで動作する．新しいデータポイント x が追加されたとする．x の最近傍データポイントを q, x が追加されたことにより x が最近傍になったデータポイント，つまり x にとっての逆最近傍データポイント集合を S とする．ステージ 1 で x に対する，q, S を求める．q, S に合わせてグラフも更新する．ステージ 2 で x のクラスタラベル L_x を更新する．ステージ 3 で連結成分になんらかの更新があったクラスタの代表点を更新し，次の階層のクラスタの更新を行う．この 3 つのステージを下の階層から順に実行することで新しく追加されたデータポイントも含めた FINCH と同じクラスタ階層が得られる．

ここで，ステージ 1 では x と直前までに追加された各データポイントとの距離を計算する必要があるため，x が追加される前までの各データポイントの個数を $n-1$ とすると $\mathcal{O}(nd)$ の時間計算量を要する．ステージ 2 では q, S の最近傍の更新により連結成分が更新されたクラスタを高々$n-1$ 個の探索で求めることができるため，$\mathcal{O}(n)$ の時間計算量を要する．ステージ 3 ではステージ 2 で探索したクラスタのラベルを保持しておくことで解決される．また前述の通り階層構造の高さは高々 $\log_2 n$ なので，1 つのデータ追加に $\mathcal{O}(nd \log n)$ の時間計算量を要する．この手順を図 2 に示した．これにより S-FINCH は $\mathcal{O}(N^2 d \log N)$ の時間計算量を要することになり，FINCH で愚直にストリーム処理させることに比べ 時間計算量を改善できる．しかし，S-FINCH は各データの追加やクラスタが変更後の更新のときに既存の全各データポイントとの距離を計算する必要がある．これにより追加されたデータ量が増えるにつれクラスタリングに膨大な時間を要することが問題となっている．

3 提 案 手 法

本節では提案手法について概説する．提案手法は，S-FINCH のクラスタリング精度を損なわずにクラスタリング結果をより高速に計算することを目的とする．まず提案手法の基本的なアイデアについて述べ，その詳細について 3. 2 節以降で説明する．

3.1 基本アイデア

S-FINCH では新しいデータが追加された時に最近傍と逆最近傍を探索するために，これまでに追加された全てのデータポイントとの距離を計算しており，時間計算量の多くを占める．ゆえに，クラスタリングにかかる時間を短縮するためには距離

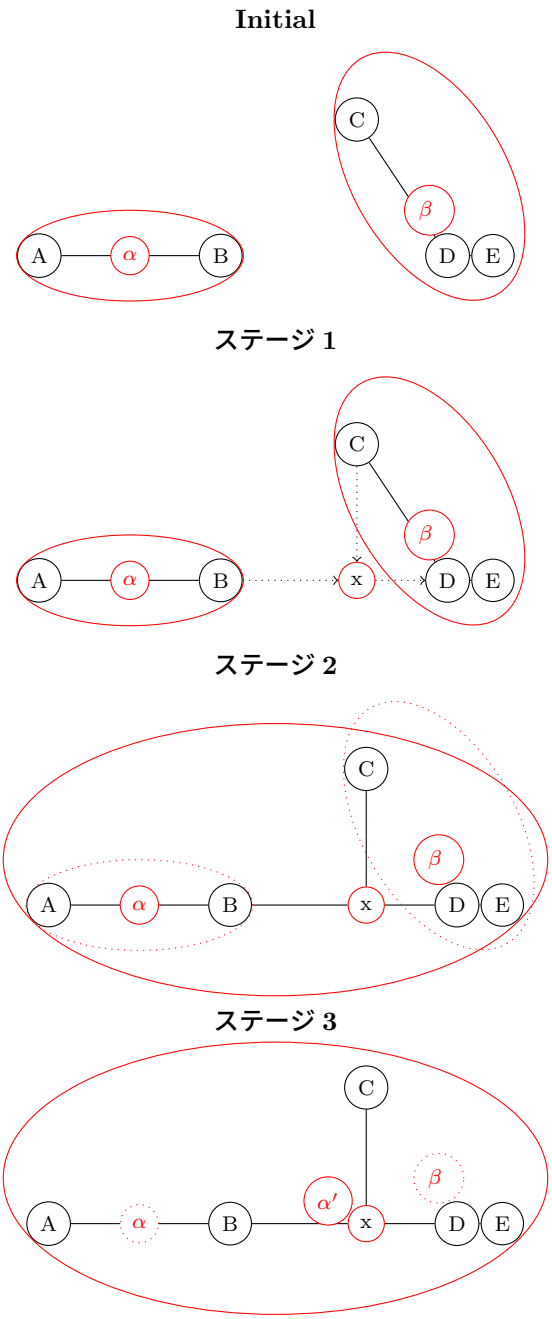


図 2: S-FINCH の詳細

計算回数を減らすことが重要である．そこで本研究ではこの最近傍と被最近傍をより少ない距離計算回数で求めることを提案手法の目的とする．

提案手法の基本アイデアは前述した距離計算において空間索引を用い，距離計算が不要なデータポイントを簡単に特定することで，全体の距離計算コストを削減することである．まず，空間索引を構築する (3.2 節)．続いて空間索引を用いた最近傍探索と逆最近傍探索で距離計算コストを下げる (3.3 節)．また，構築した空間索引はストリーム処理の各データ追加に伴い最新の状態に更新され続ける．

具体的にはデータポイント x の最近傍を探索するにあたってデータポイント a,b を包含する空間 V をおく．$dist(a,x), dist(b,x) \geqq min(dist(x,V))$ であることから，V との最小距離を計算することで V が包含する各データポイントの距離の下限を求められる．これにより最近傍を計算するにあたりデータポイント x との距離が最小距離以下であるデータポイントを既に求めている場合，データポイント x と V が包含するデータポイントの距離計算を省略できることがある．

データポイント x の逆最近傍を探索するにあたってデータポイント a を中心，a の最近傍への距離を半径とする超球 W をおく．$x \in W$ であることと a の最近傍が x になることは必要十分である．また，データポイント a,b の超球 W_a, W_b を包含する空間 S があるとする．$x \notin S$ ならば，$a,b \notin S$ である．これを利用することで，データポイント x が空間 S に含まれないとき空間 S 内の各データポイントは逆最近傍にはならないことがわかり，各データポイントとの距離計算を削減できる．

提案手法は，S-FINCH と比較して，クラスタリング精度を損なわずに大規模なデータに対して高速にクラスタリングが可能であるという優位性を持つ．これは，データが大規模になるにつれ探索において削減した距離計算コストが空間索引を構築するコストを大きく上回るようになる点によるものである．

3.2 空間索引構築

3.1 節 で述べた空間索引は，空間との最小距離を求めることと空間への包含の有無を判定する必要がある．これを実現するための空間として中心 p 半径 r の超球 S を採用する．超球を用いることでデータポイント x と空間との最小距離は $min\_dist(x,S) = max(dist(x,p) - r, 0)$ で求められ，S が x を内包するかの判定は $dist(x,p) \leqq r$ であるかどうかを確認することで実行できる．中心との距離計算は $\mathcal{O}(d)$ の時間計算量で求められるため，空間との最小距離，包含の有無いずれも $\mathcal{O}(d)$ の時間計算量しか必要としない．超矩形など異なる空間を表現するのに比べ時間計算量が小さく，効率的に空間索引の構築・利用ができる．1 つの空間索引を構築するための時間計算量と距離計算の時間計算量が同じ程度であるため，空間索引を 1 度用いるごとに 2 つ以上の頂点との距離計算が不要である判定を行うことが出来た場合に空間索引によって距離計算コストを削減出来る．また，最近傍探索と逆最近傍探索は同じ空間索引を使えないため 3.2.1 節 と 3.2.2 節 で説明する．

3.2.1 最近傍探索

3.1 節 で述べた通り，提案手法は最近傍探索のための空間索引として データポイントを包含する超球を求める．高さ h のデータポイント x の子孫のうち，高さ sh にあるデータポイントの全てを包含する超球を $V_{h,sh,x}$ とする．$h = sh$ の時， $V_{h,sh,x}$ は データポイント x の点となる．また，$\forall y \in children(x), V_{h-1,sh,y} \subset V_{h,sh,x}$ と定義できる．

また，x の全ての子 $children(x)$ の超球を包含する超球 $V_{h,sh,x}$ は中心を代表点とし，半径を $\max\{\max(dist(V_{h,sh,x}\text{の代表点}, V_{h-1,sh,y})) \mid y \in children(x)\}$ とすることで求められる．

3.2.2 逆最近傍探索

逆最近傍探索のための空間索引も 3.2.1 節 と同様に構築できる．最近傍探索と異なる点は，データポイントではなく，最

近傍距離を半径とする超球を包含する空間索引を構築することである．

高さ sh にあるデータポイント y を中心とし，y の最近傍距離を半径とする超球を $W_{sh,y}$ とする．高さ h のデータポイント x の子孫のうち，高さ sh にあるデータポイントを包含する超球を $S_{h,sh,x}$ とする．$h = sh$ の時，$S_{h,sh,x}$ は $W_{sh,x}$ となる．また，$\forall y \in children(x), S_{h-1,sh,y} \subset S_{h,sh,x}$ と定義できる．

また，x の全ての子 $children(x)$ の超球を包含する超球 $S_{h,sh,x}$ は中心を代表点とし，半径を $\max\{\max(dist(S_{h,sh,x}\text{の代表点}, S_{h-1,sh,y})) \mid y \in children(x)\}$ とすることで求められる．

3.3 探　　索

3.3.1 最近傍探索

最近傍探索は，最近傍を探索したい起点データポイント $data$，起点データポイントの存在する高さ $data\_h$ を入力として要求し，$data\_h$ における，$data$ の最近傍 $NN(data)$ を出力する．最近傍探索は，次の処理を再帰的に繰り返すことで実現される．探索する高さを $height$，起点データポイントを $data$，起点データポイントの存在する高さを $data\_h$，探索するクラスタ集合を $clusters$ とおく．暫定最近傍とは，その時点までに探索したデータポイントの中で最も $data$ に近いデータポイントのことを指し，暫定最近傍より $data$ に近いデータポイント new が見つかった場合に new が暫定最近傍へと更新される．

$height$ が $data\_h$ よりも高い場合，2 つのステージで動作する．ステージ 1 で，高さ $height$ の $clusters$ に含まれる各データポイントの超球との最小距離を計算する．ステージ 2 で，最小距離の小さいクラスタ $child$ から順に，クラスタ $child$ の子のデータポイントが暫定最近傍を更新する可能性があるか判定する．更新する可能性がある場合，次に探索する高さを $height - 1$，起点データポイントを $data$，起点データポイントの存在する高さを $data\_h$，次に探索するクラスタ集合を $children(child)$ として次の処理を実行し，返されるデータポイント x が暫定最近傍より近い場合，暫定最近傍を x で更新する．更新する可能性がない場合，残りのクラスタについては無視をする．この時点での暫定最近傍を最近傍とする．また，暫定最近傍が空であるとき，暫定最近傍との距離を ∞ とする．

$height$ が $data\_h$ と同じである場合，$height$ の $clusters$ に含まれるデータポイント x が暫定最近傍を更新するようであれば，暫定最近傍を x で更新する．全てのデータポイントについての処理後の暫定最近傍を返り値とする．

また，この処理における “クラスタ $child$ の子のデータポイントが暫定最近傍を更新する可能性” は次のように判定する．$child$ は，$height$ にある子孫に対して最近傍探索するための超球 $V_{height,data\_h,child}$ を持つため，この超球との最小距離 $min(dist(data, V_{height,data\_h,child}))$ が暫定最近傍との距離より小さい場合に更新する可能性があると判定する．以上の処理を，Algorithm 1 で示した．

Algorithm 1 に疑似コードを示す．最近傍探索の開始時には，

Algorithm 1: 最近傍探索

Input: 起点データポイント $data$，起点データポイントの存在する高さ $data\_h$，木の高さ H
Output: $data$ の最近傍

```
1  Func FIND_NN(d, d_h, h, cand, c_set):
2      distance_list ⟵ empty array
3      foreach c ∈ c_set do
4          distance_list ← distance_list.append
5          (pair(min_dist(d, V_{h,d_h,c}), c))
6      distance_list ← sort(distance_list)
7      foreach (distance, c) ∈ distance_list do
8          if distance ≧ dist(d, cand) then
9              break
10         else
11             if d_h < h then
12                 cand ←
                     FIND_NN(d, d_h, h − 1, cand, children(c))
13             else
14                 cand ← c
15     return cand
16 Func FIND_NN_entry(data, data_h):
17     return FIND_NN((data, data_h, H, null, children(root)))
```

最近傍を探索したいデータポイントを x，x の存在する高さを x_h として処理を開始する．まず，前述の入力で `FIND_NN_entry` 関数を呼び出す (line 16)．最上層のデータポイント集合が探索開始条件が設定され探索が開始される (line 1)．探索対象の各データポイントとの距離を格納するリストを初期化する (line 2)．探索対象の各データポイントとの距離を計算し (line 3)．データポイントとともに格納する (line 5)．リストを距離の昇順にソートする (line 6)．リストの先頭から確認し，暫定最近傍を更新する可能性があるかを判定する (line 8)．可能性がない場合はこのリストの処理を終了する (line 9)．探索中の階層が探索対象の階層でない場合は，データポイントの子の探索を行う (line 12)．探索中の階層が探索対象の階層である場合は，暫定最近傍の更新を行う (line 14)．リストの処理が終了した場合，暫定最近傍を返り値とする (line 15)．

3.3.2 逆最近傍探索

逆最近傍探索は，逆最近傍を探索したい起点データポイント $data$，起点データポイントの存在する高さ $data\_h$ を入力とし要求し，$data\_h$ における，$data$ の逆最近傍集合 $NNset(data)$ を出力する．逆最近傍探索は，次の処理を再帰的に繰り返すことで実現される．また，逆最近傍集合は各再帰において共通の集合である．探索する高さを $height$，起点データポイントを $data$，起点データポイントの存在する高さを $data\_h$，探索するクラスタ集合を $clusters$ とおく．

$height$ が $data\_h$ よりも高い場合，2 つのステージで動作する．$clusters$ の構成要素となる各クラスタ $child$ の子のデータポイントが逆最近傍を更新する可能性があるか判定する．可能性がある場合，次に探索する高さを $height - 1$，起点データポイント $data$，起点データポイントの存在する高さ $data\_h$，次

Algorithm 2: 逆最近傍探索

Input: 起点データポイント $data$，起点データポイントの存在する高さ $data\_h$， 木の高さ H
Output: $data$ の逆最近傍

```
1  Func FIND_NNset(d, d_h, h, cands, c_set):
2      foreach c ∈ c_set do
3          if d ∈ W_{h,d_h,c} then
4              if d_h < h then
5                  cands ← cands.append(
6                  FIND_NNset(d, d_h, h−1, cands, children(c)))
7              else
8                  cands ← c
9      return cands
10 Func FIND_NNset_entry(data, data_h):
11     return
       FIND_NNset((data, data_h, H, null, children(root)))
```

に探索するクラスタ集合 $children(child)$ として次の処理を行なう．可能性がない場合，次のクラスタへと処理をすすめる．

$height$ が $data\_h$ と同じである場合，$height$ の $clusters$ に含まれるデータポイント x 最近傍が $data$ になるようであれば，逆最近傍集合に x を追加する．

また，この処理における “各クラスタ $child$ の子のデータポイントが逆最近傍を更新する可能性” は次のように判定する．$child$ は，$height$ にある子孫に対して逆最近傍探索するための超球 $S_{height,data\_h,child}$ を持つため，この超球に $data$ が含まれる場合，逆最近傍集合を更新する可能性があると判定する．以上の処理を，Algorithm 2 で示した．

Algorithm 2 に疑似コードを示す．被最近傍探索の開始時には，被最近傍を探索したいデータポイントを x，x の存在する高さを x_h として処理を開始する．まず，前述の入力で `FIND_NNset_entry` の関数を呼び出す (line 10)．最上層のデータポイント集合が探索開始条件が設定され探索が開始される (line 1)．各データポイントが被最近傍になる可能性があるかを判定する (line 2)．可能性がない場合は次のデータポイントに進む．可能性があり，探索中の階層が探索対象の階層でない場合は，データポイントの子の探索を行い，返り値を被最近傍に追加する (line 6)．可能性があり，探索中の階層が探索対象の階層である場合は，被最近傍の追加を行う (line 7)．リストの処理が終了した場合，被最近傍を返り値とする (line 9)．

4 評価実験

提案手法の有効性を評価するために，我々の提案した高速化手法および S-FINCH に対し，処理の高速性およびクラスタリング結果の正確性の観点から比較評価を行う．

4.1 実験設定

4.1.1 実験環境

本実験には CPU Intel Xeon E5-2690 2.6 GHz, メモリ 128 GB の Linux サーバを利用する．実験に利用したプログラムは C++ で実装し，g++ (GCC) 9.2.0 で実行速度の最適化を目的に `O2` オプションをつけコンパイルした．

表 2: 実験に用いたデータセット

	N	d	計測範囲
Mice Protein	1,077	77	-
MNIST	10,000	784	-
3D Road Network	434,874	3	$[399900, 400000)$

4.1.2 データセット

本実験では，FINCH の評価実験でも利用されていたタンパク質の出現率と，8 種類の遺伝子型への分類である Mice Protein [8] ，手書き数字分類として有名な MNIST [9] で実験を行った．また新たに，各道路を構成する座標の集合である 3D Road Network [10] でも実験を行った．3D Road Network は道路 ID, 緯度, 経度, 高度 の情報を持つため，道路 ID を除く緯度, 経度, 高度 の 3 次元を入力として実験を行った．各データセットの詳細は表 2 に示す．

本実験では，各データセットをデータストリームであると想定し，1 件ずつ逐次的にデータを追加した際の，データポイント間距離計算を行った回数および実行時間の比較を行う．また，3D Road Network のように長大なデータセットにおいて実行が終了しなかったため，データセットの終盤における 100 件のデータを対象に評価を行った．対象としてデータの区間に関しては表 2 に示した．また，3D Road Network では全ての階層に対して空間索引を構築すると性能が悪化していたため階層 1,2 のみに対して空間索引を構築し階層 3 より上の階層に関しては従来手法と同じ全点への愚直な距離計算を行った．

4.2 高速性

本実験では，各データセットをデータストリームであると想定し逐次的にデータを追加し，データポイント間距離計算を行った回数の比較及び総実行時間の比較を行う．

距離計算実行回数の実験結果を図 3 と 4 に示す．横軸がそのデータが追加された時点でのデータストリームの長さであり，縦軸がそれまでに要した累計データポイント間距離計算回数とその内訳である．Mice Protein, MNIST 共に探索および空間索引構築および最近傍探索のための距離計算回数は少なくなっている．

3D Road Network の実験結果を 図 5 と 6 に示した．また，3D Road Network は，膨大な実行時間を必要としたため，10k 件で実行を打ち切っている．図 5 から提案手法は従来手法に比べて 28%ほど高速に実行可能であることが分かる．図 6 から提案手法は空間索引を構築した階層においては距離計算を 90% 以上削減，実行時間に関しては 70%削減を達成した．

4.3 正確性

提案手法と従来手法 S-FINCH の出力するクラスタリング結果の正確性について評価を行う．

提案手法は S-FINCH における最近傍と逆最近傍を探索する

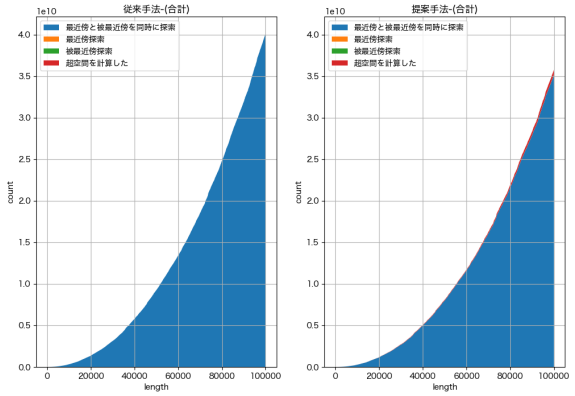


図 3: MNIST のデータポイント間距離計算の実行回数の比較

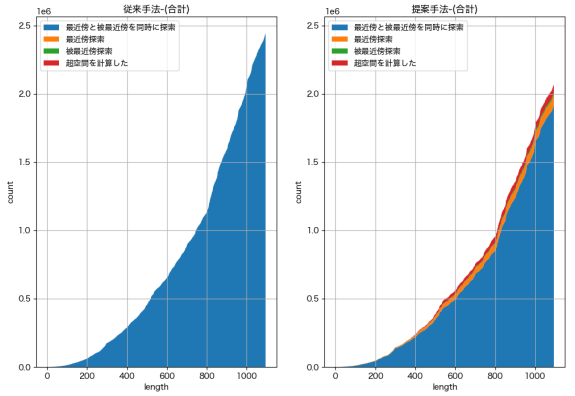


図 4: Mice Protein のデータポイント間距離計算の実行回数の比較

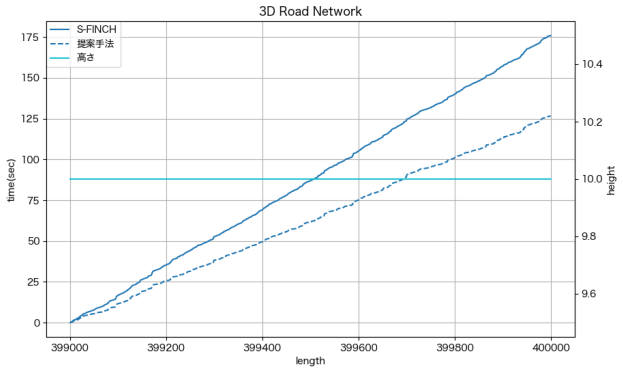


図 5: 3D Road Network の区間内での累計総実行時間と階層の高さ

部分について空間索引を導入する手法であり，この空間索引は愚直に計算を行ったときと同一の厳密解を出力するできる．そのため，提案手法は S-FINCH と完全に一致するクラスタリング結果を出力する手法となり，各データポイントが追加された後に出力される階層的クラスタリングの結果が一致する．この性質を実験的に検証するために，実行終了後における S-FINCH

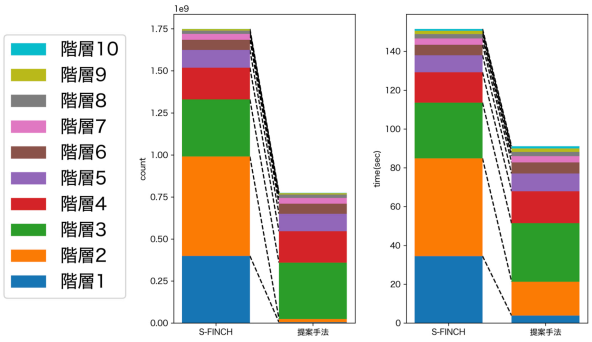


図 6: 3D Road Network の区間内での階層別の距離計算回数 (左) と探索時間 (右)

表 3: 各データセットを S-FINCH と提案手法でクラスタリングした結果の各階層おける NMI スコア

階層 (高さ)	Mice Protein	MNIST	3D Road Network
1	1.000	1.000	1.000
2	1.000	0.999	1.000
3	1.000	1.000	0.999
4	1.000	1.000	0.999
5	1.000	1.000	0.999
6	0.999	1.000	1.000
7	-	1.000	1.000
8	-	-	0.999
9	-	-	1.000

と提案手法のクラスタリング結果を比較する．評価指標として Normalized Mutual Information (NMI) を採用し，評価時には scikit-learn というライブラリに収録されている NMI 関数 [11] を利用して計算した．これによって計算される NMI スコアは 1 に近いほど性能が良く，1 は完全に一致していることを示す．その結果の NMI スコアを表 3 に示す．これは各手法のクラスタリング結果を NMI スコアを階層ごとに計算したものである．- は，そのデータセットのクラスタリング結果に存在しない階層である．その結果，全ての階層で 1.0 に近い値を出力した．しかし，1.0 を超えているものや，1.0 に限りなく近いものの 1.0 ではないもの結果も存在した．1.0 を超えることはないため，実行上の誤差ではないかと推測されるが，確証に至る根拠はない．そのため，クラスタラベルの出力を直接比較することにする．S-FINCH と提案手法は，最近傍・逆最近傍計算以外の処理は同一のものを採用している．そのため，提案手法の最近傍・逆最近傍探索の結果が正確である場合，クラスタリング結果のクラスタラベルに関しても同一のものが出力されるはずである．これを確かめるために各データセットにおいて出力されたクラスタラベルのファイルの差分を `diff` コマンドを用いて確かめた結果，一切の差分がなかった．これによりデータセットの全てにおいて提案手法と従来手法 S-FINCH は同一のクラスタリング結果を出力していることが分かった．

5 おわりに

本稿では，大規模データストリームに対して高速に S-FINCH する手法を提案し，その概要について示した．提案手法では，FINCH のもつ最近傍グラフによるクラスタを包含する空間によって空間索引を構築し，それを用いて最近傍計算における距離計算回数を削減した．提案手法は従来手法の貪欲な最近傍計算と同じ結果を返すことから，提案手法は従来手法の FINCH で生成されるクラスタリング結果と同様の処理結果をより少ない距離計算回数で出力する．これにより，本手法は大規模なデータストリームに対してより精度を損なうことなく空間索引による最近傍探索を可能にした．また，評価実験により低次元データセットに対しては実行時間を 30%削減出来ることを確認した．

今後は，高い階層における距離計算回数の削減，高次元における距離計算回数の削減を通して全体の実行時間の削減を目指すことや，より多くの低次元データセットでの評価を行いたい．

文　献

[1] Saquib Sarfraz, Vivek Sharma, and Rainer Stiefelhagen. Efficient parameter-free clustering using first neighbor relations. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8926–8935, June 2019.

[2] James MacQueen, et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1, pp. 281–297. Oakland, CA, USA, 1967.

[3] Yan-Lin He, Lei Chen, Yuan Xu, Qun-Xiong Zhu, and Shan Lu. A new distributed echo state network integrated with an auto-encoder for dynamic soft sensing. *IEEE Transactions on Instrumentation and Measurement*, Vol. 72, pp. 1–8, 2023.

[4] Yu-Ting Chang, Qiaosong Wang, Wei-Chih Hung, Robinson Piramuthu, Yi-Hsuan Tsai, and Ming-Hsuan Yang. Weakly-supervised semantic segmentation via sub-category exploration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[5] M. Saquib Sarfraz, Naila Murray, Vivek Sharma, Ali Diba, Luc Van Gool, and Rainer Stiefelhagen. Temporally-weighted hierarchical clustering for unsupervised action segmentation. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11220–11229, 2021.

[6] James Cunningham, Jim Davis, Kyle Tarplee, and Juan Vasquez. S-finch: An optimized streaming adaptation to finch clustering. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pp. 1343–1349, Aug 2022.

[7] Saquib Sarfraz, Vivek Sharma, and Rainer Stiefelhagen. Efficient parameter-free clustering using first neighbor relations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8934–8943, 2019.

[8] Clara Higuera, Katheleen J Gardiner, and Krzysztof J Cios. Self-organizing feature maps identify proteins critical to learning in a mouse model of down syndrome. *PloS one*, Vol. 10, No. 6, p. e0129126, 2015.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.

[10] Manohar Kaul. 3D Road Network (North Jutland, Denmark). UCI Machine Learning Repository, 2013. DOI: `https://doi.org/10.24432/C5GP51`.

[11] sklearn.metrics.normalized_mutual_info_score. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html`, Date of access: 2024/01/31.

一般発表 | Track 2: ビッグデータ基盤技術・セキュリティ・プライバシ

時系列データ処理

座長:渡辺 陽介(名古屋大学)
コメンテータ:喜田 拓也(北海学園大学)
2024年3月1日(金) 10:00 〜 12:10 T2-B (オンライン（Zoom Events))

11:25 〜 11:50

[T2-B-7-05]【技術報告】東芝の時系列データ処理基盤技術(GridDB)ならびにデータ仮想化エンジン(PGSpider)

浪岡保男、服部雅一、相川 恵 (株式会社東芝)