情報処理学会論文誌 Vol.61 No.2 510–519 (Feb. 2020)

ストリーミング時系列データに対するディスコードモニタリング

加藤 慎也[1,a)] 天方 大地[1,b)] 西尾 俊哉[1,c)] 原 隆浩[1,d)]

受付日 2019年6月29日, 採録日 2019年10月3日

概要：近年，多くのアプリケーションは時系列データを生成しており，その時系列データを分析することに注目が集まっている．最も重要な時系列データの分析の 1 つに異常検知があり，ディスコードの発見は時系列データ中の異常なサブシーケンスの発見に対応する．ディスコードとは，時系列データの中のすべてのサブシーケンスの中で，最近傍のサブシーケンスとの距離が最大となるサブシーケンスである．時系列データは本質的に動的であるため，ストリーミング時系列データのディスコードをモニタリングすることは非常に重要である．本論文では，スライディングウィンドウ上でストリーミング時系列データのディスコードを効率的にモニタリングする SDM（Streaming Discord Monitoring）を提案する．また，SDM を近似アルゴリズムに拡張し，精度の保証をしたうえでより効率的にディスコードをモニタリングする．実データを用いた実験により，SDM およびその近似アルゴリズムの有効性を確認する．

キーワード：時系列データ，ディスコード，スライディングウィンドウ

Monitoring Discord on Streaming Time-series

Shinya Kato[1,a)] Daichi Amagata[1,b)] Shunya Nishio[1,c)] Takahiro Hara[1,d)]

Received: June 29, 2019, Accepted: October 3, 2019

Abstract: Many applications generate time-series, and time-series analysis has been receiving many attentions. One of the most important time-series analysis tools is anomaly detection, and discord discovery aims at finding an anomaly subsequence in a time-series. Given a time-series, the discord of the time-series is the subsequence with the largest distance to its nearest neighbor among all subsequences. Time-series is essentially dynamic, so monitoring the discord of a streaming time-series is an important problem. This paper addresses this problem and proposes SDM (Streaming Discord Monitoring), an algorithm that efficiently updates the discord of a streaming time-series over a sliding window. We show that SDM is approximation-friendly, i.e., the computational efficiency is accelerated by monitoring an approximate discord with theoretical bound. Our experiments on real datasets demonstrate the efficiency of SDM and its approximate version.

Keywords: time-series, discord, sliding window

1. 序論

研究動機．実世界の多くのアプリケーションは時系列データを生成しており，有用な知識を得るためにその時系列データを利用している [6]．異常検知は，有用な知識を得るためのツールの 1 つであり，異常な観測値の発見やデータクリーニングを可能とする [21], [25]．時系列データの異常検知の最も効果的な方法の 1 つとしてディスコードがある [11], [12], [23]．時系列データのディスコードとは，その時系列データの中のすべてのサブシーケンスに対して，最近傍のサブシーケンスとの距離が最大となるサブシーケンスである（正確な定義は 2 章で紹介する）．たとえば，**図 1** は心電図の時系列データを表しており，青いサブシーケンスがディスコードである．ディスコードの発見は産業 [3],

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[a)] kato.shinya@ist.osaka-u.ac.jp
[b)] amagata.daichi@ist.osaka-u.ac.jp
[c)] nishio.syunya@ist.osaka-u.ac.jp
[d)] hara@ist.osaka-u.ac.jp

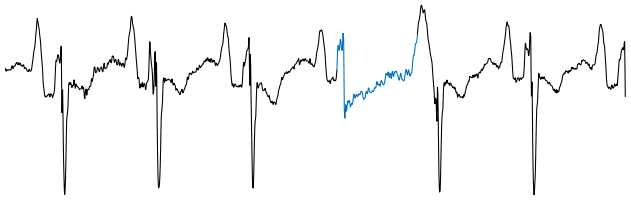


図 1 心電図の時系列データおよびそのディスコード
Fig. 1 ECG time-series and its discord (the blue subsequence).

医療 [22]，および Web [23] において有用であることが示されている．これらのアプリケーションではストリーミング時系列データを生成しており [13]，ディスコードは時間の経過とともに変化する．そのため，ストリーミング時系列データのディスコードをリアルタイムにモニタリングすることは重要な問題である．そこで，本論文では，カウントベースのスライディングウィンドウ上でストリーミング時系列データのディスコードをモニタリングする問題に取り組む．

課題． ウィンドウがスライドした際，新たな値がウィンドウに挿入され，最も古い値がウィンドウから削除される．つまり，新たな値を含むサブシーケンス s_n が生成され，最も古い値を含むサブシーケンス s_e が削除される．このとき，s_n および s_e により，ウィンドウ内のサブシーケンスの最近傍が変化し，ディスコードが変化する可能性がある．先述したアプリケーションでは，リアルタイムにディスコードをモニタリングする必要があるため，ディスコードが変化するかどうかを高速に把握する必要がある．ディスコードをモニタリングする最も単純な手法として，ウィンドウがスライドした際，ウィンドウ内のすべてのサブシーケンスに対して最近傍探索を実行することが考えられる．しかし，この手法はディスコードの更新に多大な時間がかかり，リアルタイム性を保証することが難しい．既存のディスコードを発見するアルゴリズム [11] を用いてディスコードをモニタリングすることはできるが，ディスコードが削除された際，ウィンドウ内のすべてのサブシーケンスに対して最近傍探索を実行する必要があり，リアルタイムモニタリングに適していない．

以上の議論から，効率的にディスコードをモニタリングするためには，(i) ディスコードが変化するかどうかを高速に把握し，(ii) ディスコードが変化した際，高速にディスコードを発見する必要がある．

提案アルゴリズムの概要． この問題を解決するため，カウントベースのスライディングウィンドウ上でディスコードを効率的にモニタリングするアルゴリズム SDM（**S**treaming **D**iscord **M**onitoring）を提案する．SDM は，高次元データに対して高速であるシーケンシャルスキャンによる最近傍探索を用いる [20]．このアプローチは，最近傍が変化するサブシーケンスを効率的に特定できるため，課題 (i) を解決できる．また，ウィンドウ内の各サブシーケンスが 2 種類の最近傍を保持することで，ディスコードが変化した際に必要となる計算を枝刈りできる．つまり，課題 (ii) を解決できる．さらに，SDM は，ディスコードの精度を保証する近似アルゴリズムに拡張でき，最悪更新時間を短縮できる．

貢献． 以下に本研究の貢献を示す．

- スライディングウィンドウ上でストリーミング時系列データのディスコードをモニタリングする問題に取り組む．筆者らの知る限り，この問題はこれまでに取り組まれていない．
- 効率的に本問題を解決するアルゴリズム SDM を提案する．
- SDM の近似アルゴリズム A-SDM を提案する．A-SDM は，ディスコードの精度の保証をしたうえで，最悪更新時間を短縮する．
- 実データを用いた実験により，SDM および A-SDM が他のアルゴリズムよりも高速であることを示す．

本論文の構成． 2 章で本論文の問題を定義し，3 章で関連研究について述べる．4 章で SDM について説明し，5 章で実データを用いた実験の結果を示す．最後に 6 章で本論文のまとめと今後の課題について述べる．

2. 問題定義

ストリーミング時系列データ t は実数値の系列であり，$t = (t[1], t[2], \ldots)$ と表現する．まず，t の一部を表すサブシーケンスを定義する．

定義 1（サブシーケンス）． t および長さ l が与えられたとき，p 番目の値を始点とする t のサブシーケンス s_p は式 (1) により定義される．

$$s_p = (t[p], t[p+1], \ldots, t[p+l-1]) \tag{1}$$

ここで，s_p の x 番目のデータの値を $s_p[x]$ と表現する．つまり，$s_p = (s_p[1], s_p[2], \ldots, s_p[l])$ である．次に，t の中で s_p の最近傍を計算するため，本研究では，時系列データ間の距離を測る基本的な指標である z 正規化ユークリッド距離を用いる [17], [24]．

定義 2（z 正規化ユークリッド距離）． 長さ l の 2 つのサブシーケンス s_p および s_q が与えられたとき，これらの z 正規化ユークリッド距離 $dist(s_p, s_q)$ は式 (2) により定義される．

$$dist(s_p, s_q) = \sqrt{\sum_{i=1}^{l} \left(\frac{s_p[i] - \mu(s_p)}{\sigma(s_p)} - \frac{s_q[i] - \mu(s_q)}{\sigma(s_q)} \right)^2} \tag{2}$$

ここで，$\mu(s)$ および $\sigma(s)$ はそれぞれ $(s[1], s[2], \ldots, s[l])$ の平均および標準偏差である（z 正規化ユークリッド距離の時間計算量は $\mathcal{O}(l)$ である）．

次に，s_p と s_{p+1} が互いに類似していることは自明であり，有用な結果を得るためには，このようなサブシーケンスを考慮すべきではない [2], [15]．そこで，互いに重なり合うサブシーケンスをトリビアルマッチと定義する．

定義 3（トリビアルマッチ）[5]. s_p が与えられたとき，s_p とトリビアルマッチであるサブシーケンスの集合 S_p は次の条件を満たす．

$$S_p = \{s_q | p - l + 1 \leq q \leq p + l - 1\} \tag{3}$$

ここで，1 章で述べたアプリケーションを含む多くのアプリケーションでは最新の値のみを考慮している [14]．そのため，ストリーミング時系列データに関する既存の研究 [9], [10], [13] と同様に，本研究においてもカウントベースのスライディングウィンドウを用いて，最新の w 個の値のみをモニタリングする．つまり，ウィンドウ内のストリーミング時系列データ t は $t = (t[i], t[i+1], \ldots, t[i+w-1])$ のように表され，$t[i+w-1]$ が最新の値である．また，l が与えられたとき，ウィンドウ内には $w - l + 1$ 個のサブシーケンスが含まれる．ウィンドウがスライドしたとき，最新の l 個の値を含む新たなサブシーケンスが生成される．同時に，最も古い値がウィンドウから削除されるため，最も古いサブシーケンスが削除される．また，ウィンドウ内のすべてのサブシーケンスはメモリ上にあると想定する．

本研究では，このような環境において最近傍との距離が最大となるサブシーケンスをモニタリングする．ここで，サブシーケンスの最近傍を定義する．

定義 4（最近傍）. s_p およびウィンドウ内のすべてのサブシーケンスの集合 S が与えられたとき，s_p の最近傍 s_q は次の条件を満たす．

$$\arg\min_{s_q \in S \setminus S_p} dist(s_p, s_q) \tag{4}$$

s_p と s_p の最近傍との距離を $s_p.dist_{NN}$ としたとき，本研究の問題は以下のように定義される．

問題定義. t，w，および l が与えられたとき，式 (5) で表されるディスコード s^* をモニタリングする．

$$s^* = \arg\max_{s_p \in S} s_p.dist_{NN} \tag{5}$$

ディスコードモニタリングを用いるアプリケーションは，ディスコードをリアルタイムに更新する必要がある．そのため，s^* の更新時間の高速化を図る．

3. 関連研究

時系列データマイニングに関する研究は多く行われている [1], [6], [7]．本章では，本研究に最も関連しているディスコードの発見およびモニタリングに関する既存研究についてのみ紹介する．

静的な時系列データに対するディスコードの発見. 文献 [8], [11], [23] では，静的な時系列データに対して効率的にディスコードを発見するアルゴリズムを提案している．文献 [11] では，メモリ上の時系列データに対してディスコードを効率的に発見するアルゴリズム HOT SAX を提案している．HOT SAX では，各サブシーケンス s_p の最近傍を計算をするとき，s_p の最近傍との距離の上界値がディスコードの最近傍との距離より小さくなった場合，以降の計算を打ち切る．計算を効率的に打ち切るため，HOT SAX では，SAX（Symbolic Aggregate Approximation）[16] によりサブシーケンスをシンボル（記号列）に変換し，シンボルが類似するサブシーケンス間の距離は小さくなるという傾向を用いる．そのため，シンボルが類似するサブシーケンス間の距離計算を初期の試行で行う．HOT SAX を本研究の問題に用いることはできるが，ディスコードが削除されたときに再計算が必要であり，多大な更新時間がかかる．これを 5 章で示す．

文献 [23] では，時系列データがディスク上にあると想定し，SDM とは異なるシーケンシャルスキャンに基づくアルゴリズムを提案している．文献 [8] では，ディスコードの発見を並列化することで高速化するアルゴリズムを提案している．本論文では，並列計算は対象外である．

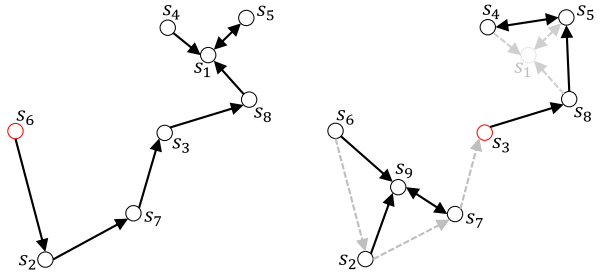
動的な時系列データに対するディスコードのモニタリング. 文献 [19], [24] では，ストリーミング時系列データに対してディスコードをモニタリングするアルゴリズムを提案している．文献 [19] では，新たなサブシーケンスがディスコードになるかどうかを高速に把握するため R 木を用いる．一方，文献 [24] では，静的な時系列データの発見に用いられているデータ構造である Matrix Profile を用いる．これらの研究は値の追加にのみ対応しており，値の削除には対応していない．

4. SDM: Streaming Discord Monitoring

ウィンドウがスライドした際，新たな値がウィンドウに挿入され，最も古い値がウィンドウから削除される．つまり，新たな値を含むサブシーケンス s_n が生成され，最も古い値を含むサブシーケンス s_e が削除される．これにより，以下の場合にディスコードが変化する可能性がある．

- s_n がディスコードになる．
- s_e を最近傍としていたサブシーケンスがディスコードになる．
- s_e がディスコードであった．
- s_n があるサブシーケンスの最近傍となることでディスコードが変化する．

例 1. **図 2** に $w = 9$ のスライディングウィンドウ上のストリーミング時系列データを示す．$l = 2$ とし，各サブシーケンスを 2 次元上の点として表現している．また，矢印の向きは最近傍であることを表し（たとえば，s_1 の最近傍は s_5 である），矢印の長さは最近傍との距離を表現してい



(a) s_9 生成前　(b) s_9 生成後

図 2　ウィンドウのスライドによるディスコードの変化
Fig. 2　Discord update due to window sliding.

る．図 2 (a) では s_6 がディスコードであり，図 2 (b) では，ウィンドウのスライドにより，s_1 が削除され，s_9 が生成されている．s_1 の削除および s_9 の生成により，s_4，s_5，s_6，s_7，および s_8 の最近傍が変化している．その結果，ディスコードが s_6 から s_3 に更新される．

4.1　メインアイデア

SDM は，シンプルなデータ構造およびシーケンシャルスキャンにより効率的にディスコードをモニタリングする．具体的には，以下のアイデアから問題を解決する．

(1) 各サブシーケンスの最近傍および最近傍との距離を保持することにより，容易にディスコードを特定できる．
(2) すべてのサブシーケンスをスキャンすることで s_n の最近傍が計算でき，他のサブシーケンスの最近傍を逐次更新できる．また，ウィンドウがスライドする前に s_e を最近傍としていたサブシーケンスを特定できる．

スキャンベースの手法 (2) は単純であるが，ディスコードのモニタリングには効果的である．高速に最近傍を計算するために，HOT SAX などのインデックスを用いた手法も考えられる．しかし，枝刈りされたサブシーケンスの最近傍を更新できず，ディスコードを高速に更新できない．5 章において，このような手法の最悪更新時間が大きくなり，リアルタイムモニタリングに適さないことを示す．

4.2　データ構造

SDM では，各サブシーケンスは $\mathrm{NN}_{\mathrm{older}}$-タプル，$\mathrm{NN}_{\mathrm{younger}}$-タプル，および NN-タプルを保持する．

定義 5（$\mathrm{NN}_{\mathrm{older}}$-タプル）． s_p の $\mathrm{NN}_{\mathrm{older}}$-タプルは以下のペアで構成される．

- $s_p.id_{NN_{older}}$：s_p より前に生成されたサブシーケンスの集合における s_p の最近傍の識別子
- $s_p.dist_{NN_{older}}$：s_p より前に生成されたサブシーケンスの集合における s_p の最近傍との距離

定義 6（$\mathrm{NN}_{\mathrm{younger}}$-タプル）． s_p の $\mathrm{NN}_{\mathrm{younger}}$-タプルは以下のペアで構成される．

- $s_p.id_{NN_{younger}}$：s_p より後に生成されたサブシーケンスの集合における s_p の最近傍の識別子

サブシーケンス	$\mathrm{NN}_{\mathrm{older}}$-タプル	$\mathrm{NN}_{\mathrm{younger}}$-タプル	NN-タプル
s_1	-	⟨5,1.0⟩	⟨5,1.0⟩
s_2	-	⟨7,2.7⟩	⟨7,2.7⟩
s_3	-	⟨8,2.5⟩	⟨8,2.5⟩
s_4	⟨1,0.8⟩	⟨5,1.5⟩	⟨1,0.8⟩
s_5	⟨1,1.0⟩	⟨8,1.8⟩	⟨1,1.0⟩
s_6	⟨2,3.0⟩	⟨7,3.1⟩	⟨2,3.0⟩
s_7	⟨3,2.2⟩	⟨8,3.2⟩	⟨3,2.2⟩
s_8	⟨1,1.4⟩	-	⟨1,1.4⟩

(a) s_9 生成前

サブシーケンス	$\mathrm{NN}_{\mathrm{older}}$-タプル	$\mathrm{NN}_{\mathrm{younger}}$-タプル	NN-タプル
s_2	-	⟨9,1.5⟩	⟨9,1.5⟩
s_3	-	⟨8,2.5⟩	⟨8,2.5⟩
s_4	-	⟨5,1.5⟩	⟨5,1.5⟩
s_5	⟨4,1.3⟩	⟨8,1.8⟩	⟨4,1.3⟩
s_6	-	⟨9,2.4⟩	⟨9,2.4⟩
s_7	-	⟨9,1.2⟩	⟨9,1.2⟩
s_8	⟨5,1.6⟩	⟨9,4.2⟩	⟨5,1.6⟩
s_9	⟨7,1.2⟩	-	⟨7,1.2⟩

(b) s_9 生成後

図 3　図 2 の各サブシーケンスのデータ構造
Fig. 3　Data structure of each subsequence in Fig. 2.

- $s_p.dist_{NN_{younger}}$：s_p より後に生成されたサブシーケンスの集合における s_p の最近傍との距離

定義 7（NN-タプル）． s_p の NN-タプルは s_p の最近傍の識別子および最近傍との距離により構成される．$s_p.dist_{NN_{older}} < s_p.dist_{NN_{younger}}$ である場合，s_p の NN-タプルは $\mathrm{NN}_{\mathrm{older}}$-タプルである．そうでない場合，$s_p$ の NN-タプルは $\mathrm{NN}_{\mathrm{younger}}$-タプルである．さらに，$s_p$ の NN-タプルが $\mathrm{NN}_{\mathrm{younger}}$-タプルであるとき，つまり，$s_p.dist_{NN_{older}} \geq s_p.dist_{NN_{younger}}$ であるとき，$\mathrm{NN}_{\mathrm{older}}$-タプルは NN-タプルになり得ないため，$\mathrm{NN}_{\mathrm{older}}$-タプルを保持する必要はない．この性質を利用して不必要な距離計算を削減できる．

例 2． 図 3 に，図 2 の各サブシーケンスの $\mathrm{NN}_{\mathrm{older}}$-タプル，$\mathrm{NN}_{\mathrm{younger}}$-タプル，および NN-タプルを示す．図 3 (a) において，s_2 の $\mathrm{NN}_{\mathrm{older}}$-タプルが空であるのは，$s_2.dist_{NN_{older}} > s_2.dist_{NN_{younger}}$ であるからである．図 3 (b) において赤字の部分は s_1 の削除および s_9 の生成により図 3 (a) から変化した部分である．

$\mathrm{NN}_{\mathrm{older}}$-タプルと $\mathrm{NN}_{\mathrm{younger}}$-タプルを保持する理由． $s_q.id_{NN_{older}} = p$，$s_q.id_{NN_{younger}} = r$ であると仮定したとき，s_q が削除される前に s_p が削除される．s_p が削除されるとき，$dist(s_p, s_q) < dist(s_q, s_r)$ である場合，s_q の最近傍は分からなくなる（$dist(s_p, s_q) \geq dist(s_q, s_r)$ である場合，s_q の最近傍は s_r である）．しかし，$s_q.dist_{NN_{younger}} =$

$dist(s_q, s_r)$ が $s^*.dist_{NN}$ より小さい場合，s_q がディスコードにならないことが分かる．つまり，s_q に対して最近傍探索を実行する必要がないため，更新時間を削減できる．さらに，$s_q.dist_{NN_{younger}} = dist(s_q, s_r)$ が $s^*.dist_{NN}$ より大きかったとしても，s_q の $\mathrm{NN}_{\mathrm{older}}$-タプルのみを探索すればよいため，探索範囲を大幅に削減できる．一方，NN-タプルしか用いない場合，s_q の最近傍が削除されたとき，探索範囲が大きいため，大きな更新時間がかかる．

4.3 アルゴリズム

シーケンシャルスキャンを用いる理論的根拠． ウィンドウがスライドした際，ディスコードが変化するかどうかを把握する必要がある．新たに生成されるサブシーケンス s_n により，s_p の最近傍が変化する可能性がある．すべての s_p に対して s_n は最も新しいため，各サブシーケンスの $\mathrm{NN}_{\mathrm{younger}}$-タプルの更新が行われる．$l$ を固定したとき，各 s_p に対して，$\min\{s_p.dist_{NN_{younger}}, dist(s_p, s_n)\}$ をチェックするのに $\mathcal{O}(1)$ 時間かかるため，全体の時間計算量は $\mathcal{O}(w)$ である．これは，ウィンドウ内のすべてのサブシーケンスをシーケンシャルスキャンすることとまったく同じ計算量である．

アルゴリズム． SDM は，上記の分析および 4.2 節に基づいて設計されており，Algorithm 1 に詳細を示す．まず，ウィンドウがスライドする前のディスコードを s^*_{temp} とする．次に，ウィンドウ内のすべてのサブシーケンスの集合 S から，削除されるサブシーケンス s_e を削除し，新たに生成されるサブシーケンス s_n を追加する．次に，s_n の $\mathrm{NN}_{\mathrm{older}}$-タプル，$\mathrm{NN}_{\mathrm{younger}}$-タプル，および NN-タプルを初期化する．最後に，Nearest-Neighbor-Search を実行する．これは，ウィンドウ内のサブシーケンスの $\mathrm{NN}_{\mathrm{younger}}$-タプルを更新しながら，$s_n$ の最近傍を計算するアルゴリズムである．

Nearest-Neighbor-Search. Algorithm 2 は，最近傍探索を行うアルゴリズムを示している．$s_p \in S\backslash\{s_n\}$ が与えられたとき，以下の処理を実行する．まず，$dist(s_p, s_n)$ を計算する．必要に応じて，s_n の $\mathrm{NN}_{\mathrm{older}}$-タプル（3–4 行），および s_p の $\mathrm{NN}_{\mathrm{younger}}$-タプル（5–7 行）を更新する．$s_p$ の $\mathrm{NN}_{\mathrm{younger}}$-タプルが更新された場合，必要に応じて，$s_p$ の NN-タプルを更新する（8–10 行）．さらに，s_p の NN-タプルが更新され，s_p が s^*_{temp} である場合，s^*_{temp} の最近傍との距離が小さくなる．これにより，ディスコードが変化する可能性があるため，Discord-Update を実行する．Discord-Update は s^*_{temp} を得るアルゴリズムであり，詳細は後述する．

次に，s_p の最近傍が削除されるかどうかを確認する．s_p の最近傍が削除される場合，s_p の $\mathrm{NN}_{\mathrm{older}}$-タプルおよび NN-タプルを初期化し（12 行），s_p が s^*_{temp} である場合，Discord-Update を実行する．$s_p.dist_{NN_{younger}} \leq s^*_{temp}.dist_{NN}$ である場合，s_p は s^* にはならないため，s_p に対して最近傍探索を実行する必要はない．そうでない場合，Older-Nearest-Neighbor-Search(s_p) を実行する．これは，s_p の $\mathrm{NN}_{\mathrm{older}}$-タプルをシーケンシャルスキャンにより計算し，必要に応じて s^*_{temp} を更新するアルゴリズムである．

すべての $s_p \in S\backslash\{s_n\}$ に対して以上の操作を実行した後，s_n の NN-タプルを取得する（17 行）．最後に，s^* を取得する（18 行）．

Discord-Update. Algorithm 3 は，ディスコードの更新を行うアルゴリズムを示している．s^*_{temp} を更新する必要がある場合，s^*_{temp} の NN-タプルを初期化する（1 行）．NN-タプルが空でないサブシーケンスの集合をスキャンすることで，s^*_{temp} を取得する（2–4 行）．さらに，NN-タプルが空であるサブシーケンスの集合をスキャンし，$s_p.dist_{NN_{younger}} < s^*_{temp}.dist_{NN}$ を満たす s_p をヒープ H

Algorithm 1: SDM

Input: s_e: the expired subsequence, s_n: the new subsequence
Output: s^*: the discord

1 $s^*_{temp} \leftarrow$ the discord before the window slided
2 $S \leftarrow S\backslash\{s_e\}$, $S \leftarrow S \cup \{s_n\}$ ▷ S is the set of all subsequences on the window
3 $s_n.\langle\cdot,\cdot\rangle_{NN} \leftarrow \varnothing$, $s_n.\langle\cdot,\cdot\rangle_{NN_{older}} \leftarrow \varnothing$, $s_n.\langle\cdot,\cdot\rangle_{NN_{younger}} \leftarrow \varnothing$
4 $s^* \leftarrow$ Nearest-Neighbor-Search

Algorithm 2: Nearest-Neighbor-Search

Input: S: the set of all subsequences, s^*_{temp}: a temporal discord
Output: s^*: the discord

1 **for** $\forall s_p \in S\backslash S_n$ **do**
2 $d \leftarrow dist(s_p, s_n)$
3 **if** $s_n.dist_{NN_{older}} > d$ **then**
4 $s_n.\langle\cdot,\cdot\rangle_{NN_{older}} \leftarrow \langle p, d\rangle$
5 **if** $s_p.dist_{NN_{younger}} > d$ **then**
6 $s_p.\langle\cdot,\cdot\rangle_{NN_{younger}} \leftarrow \langle n, d\rangle$
7 **if** $s_p.\langle\cdot,\cdot\rangle \neq \varnothing \wedge s_p.dist_{NN} > s_p.dist_{NN_{younger}}$ **then**
8 $s_p.\langle\cdot,\cdot\rangle_{NN} \leftarrow s_p.\langle\cdot,\cdot\rangle_{NN_{younger}}$
9 **if** $s_p = s^*_{temp}$ **then**
10 $s^*_{temp} \leftarrow$ Discord-Update
11 **if** $s_p.id_{NN} = e$ **then**
12 $s_p.\langle\cdot,\cdot\rangle_{NN_{older}} \leftarrow \varnothing$, $s_p.\langle\cdot,\cdot\rangle_{NN} \leftarrow \varnothing$
13 **if** $s_p = s^*_{temp}$ **then**
14 $s^*_{temp} \leftarrow$ Discord-Update
15 **if** $s_p.dist_{NN_{younger}} > s^*_{temp}.dist_{NN}$ **then**
16 $s^*_{temp} \leftarrow$ Older-Nearest-Neighbor-Search(s_p)
17 $s_n.\langle\cdot,\cdot\rangle_{NN} \leftarrow s_n.\langle\cdot,\cdot\rangle_{NN_{older}}$
18 $s^* \leftarrow s^*_{temp}$

Algorithm 3: Discord-Update

```
Output: s*_temp: a temporal discord
1  s*_temp.⟨·,·⟩_NN ← ⟨−1, 0⟩
2  for ∀s_p ∈ S such that s_p.⟨·,·⟩_NN ≠ ∅ do
3      if s*_temp.dist_NN < s_p.dist_NN then
4          s*_temp ← s_p
5  for ∀s_p ∈ S such that s_p.⟨·,·⟩_NN = ∅ do
6      if s_p.dist_NN_younger > s*_temp.dist_NN then
7          Up-Max-Heap(H, s_p)        ▷ s_p is inserted into a
             heap H
8  while H ≠ ∅ do
9      s_p ← Down-Max-Heap(H)         ▷ s_p is poped from H
10     if s_p.dist_NN_younger < s*_temp.dist_NN then
11         break
12     else
13         s*_temp ← Older-Nearest-Neighbor-Search(s_p)
14 return s*_temp
```

に追加する（5–7 行）．ここで，H 内のサブシーケンスは $s_p.dist_{NN_{younger}}$ の降順にソートされている．

次に，H が空でない場合，H の先頭から 1 つサブシーケンス s_p を取り出し，s_p が $s_p.dist_{NN_{younger}} < s^*_{temp}.dist_{NN}$ を満たすかどうかを確認する（8–9 行）．満たす場合，Discord-Update を終了する（11 行）．そうでない場合，Older-Nearest-Neighbor-Search(s_p) を実行する（13 行）．H および $\mathrm{NN}_{\mathrm{older}}$-タプルを用いることにより，不必要な Older-Nearest-Neighbor-Search(·) の実行を避けることができる．

以下で，SDM の空間計算量および時間計算量について説明する．

定理 1（空間計算量）． SDM の空間計算量は $\mathcal{O}(w)$ である．

証明． SDM では，$\mathrm{NN}_{\mathrm{older}}$-タプル，$\mathrm{NN}_{\mathrm{younger}}$-タプル，および NN-タプルを保持し，また，それらは $\mathcal{O}(1)$ のメモリコストを必要とする．ウィンドウ内のサブシーケンスの数は $\mathcal{O}(w)$ であるため，SDM の空間計算量は $\mathcal{O}(w)$ である．□

定理 2（時間計算量）． SDM の時間計算量は $\mathcal{O}((1+c)wl + c'(w + h\log h))$ である．ただし，c は Older-Nearest-Neighbor-Search(·) の実行回数，c' は Discord-Update の実行回数，および h は Algorithm 3 の 8 行目のヒープのサイズである．

証明． SDM の主な計算コストは Nearest-Neighbor-Search のコストである．Nearest-Neighbor-Search は S をスキャンし s_p と s_n 間の距離を計算するため，少なくとも $\mathcal{O}(wl)$ 時間かかる．また，スキャン中，何度か Older-Nearest-Neighbor-Search(·) を実行する可能性があり，それにも $\mathcal{O}(wl)$ 時間かかる．さらに，Discord-Update に $\mathcal{O}(w + h\log h)$ 時間かかる．したがって，Nearest-Neighbor-Search には $\mathcal{O}((1+c)wl + c'(w + h\log h))$ 時間かかるため，SDM の時間計算量は $\mathcal{O}((1+c)wl + c'(w + h\log h))$ である．□

4.4 A-SDM

定理 2 より，SDM の時間計算量は Older-Nearest-Neighbor-Search(·) の実行回数および Discord-Update の実行回数に依存することが分かる（実験では，1 回のスライドにおいて通常 $c = c' = 0$ であることが分かった）．多くのサブシーケンスの最近傍が同時に削除されると c および c' が大きくなり，ディスコードの更新に大きな時間がかかる．アプリケーションが正確なディスコードを要求しない場合，c および c' を小さくすることで更新時間を削減できる．

そこで，ディスコードの精度を保証する SDM の近似アルゴリズム A-SDM（**A**pproximate **SDM**）を提案する．A-SDM は，SDM をわずかに拡張したアルゴリズムである．アプリケーションによって指定される近似係数 ϵ (> 1) が与えられたとき，Algorithm 2 の 15 行目および Algorithm 3 の 10 行目を

$$s_p.dist_{NN_{younger}} > \epsilon \cdot s^*_{temp}.dist_{NN}$$

に置き換え，Algorithm 3 の 6 行目を

$$s_p.dist_{NN_{younger}} < \epsilon \cdot s^*_{temp}.dist_{NN}$$

に置き換える．このとき，以下の定理が成り立つ．

定理 3（精度保証）． A-SDM がモニタリングしているサブシーケンスを s_{out} としたとき，以下の不等式が成り立つ．

$$s_{out}.dist_{NN} \geq \frac{s^*.dist_{NN}}{\epsilon}. \tag{6}$$

証明． まず，Discord-Update，つまり Algorithm 3 に対して不等式 (6) が成り立つことを示す．$s_p.dist_{NN_{younger}} \leq \epsilon \cdot s^*_{temp}.dist_{NN}$ である場合，s_p は H に挿入されない．$s_p.dist_{NN} \leq s_p.dist_{NN_{younger}} \leq \epsilon \cdot s^*_{temp}.dist_{NN}$ であるため，s_p が s^* であっても不等式 (6) は成り立つ．同様に，10 行目においても成り立つ．

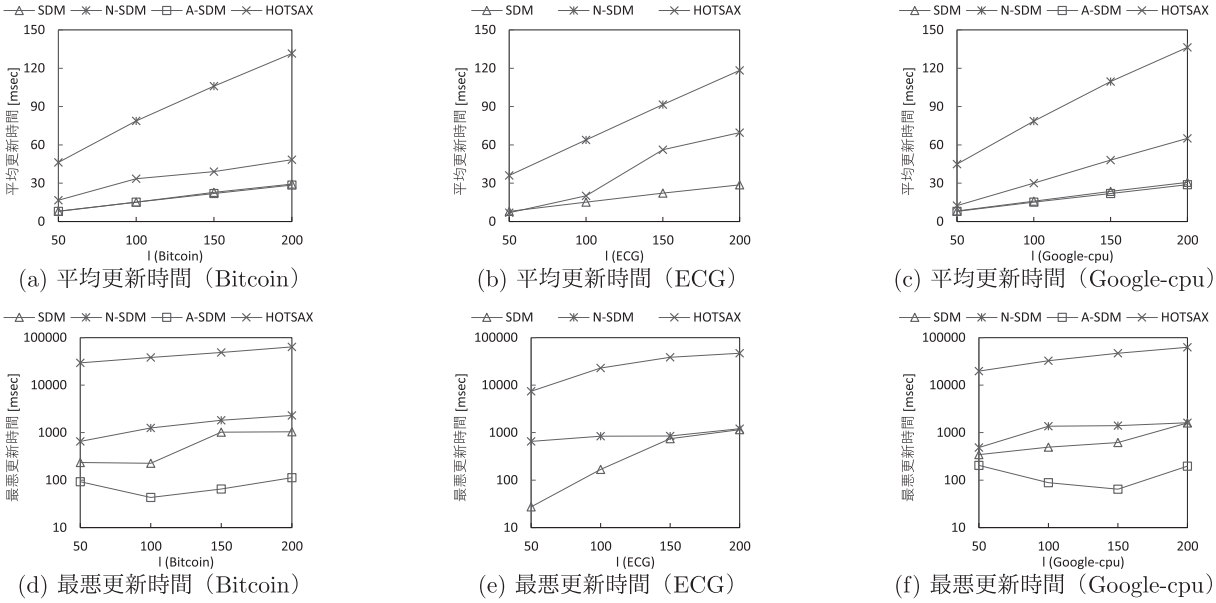
次に，Algorithm 2 の 15 行目に注目する．このとき，s^* は s_p または s^*_{temp} である．$s^*_{temp} = s^*$ である場合，A-SDM は正確なディスコードをモニタリングしているため，不等式 (6) が成り立つ．一方，$s_p = s^*$ かつ $s_p.dist_{NN_{younger}} \leq \epsilon \cdot s^*_{temp}.dist_{NN}$ である場合，A-SDM は $s^*_{temp} = s_{out}$ をモニタリングしている．これは，Discord-Update の場合と同様に不等式 (6) が成り立つ．以上により，定理 3 が成り立つ．□

5. 評価実験

すべての実験は，Windows 10 Pro for Workstations, 3.00 GHz Intel Xeon Gold，および 512 GB RAM を搭載した計算機で行った．

5.1 セッティング

データセット． 以下の 3 つの実データを用いた．



(a) 平均更新時間（Bitcoin） (b) 平均更新時間（ECG） (c) 平均更新時間（Google-cpu）

(d) 最悪更新時間（Bitcoin） (e) 最悪更新時間（ECG） (f) 最悪更新時間（Google-cpu）

図 4 l の影響

Fig. 4 Impact of l (subsequence size).

- Bitcoin[*1]：ビットコインのトランザクションのストリーミング時系列データ（長さ 100,000）
- ECG [4]：心電図のストリーミング時系列データ（長さ 100,000）
- Google-cpu [18]：Google のデータセンタの CPU 使用率のストリーミング時系列データ（長さ 133,902）

アルゴリズム．以下のアルゴリズムを C++で実装し，評価実験を行った．

- HOT SAX [11]：静的な時系列データに対するディスコード発見のアルゴリズムであり，ウィンドウのスライドによるディスコードの更新に対応できるように拡張した．
- N-SDM：NN-タプルのみを保持する SDM であり，$\mathrm{NN}_{\mathrm{older}}$-タプルおよび $\mathrm{NN}_{\mathrm{younger}}$-タプルは保持しない（N-SDM は，$\mathrm{NN}_{\mathrm{older}}$-タプルおよび $\mathrm{NN}_{\mathrm{younger}}$-タプルの性能を評価するために用いる）．
- SDM：本論文の提案アルゴリズムである．
- A-SDM：SDM の近似アルゴリズムである．

他の既存のディスコード発見/モニタリングのアルゴリズムは，サブシーケンスの削除やディスコードの削除に対応できないため比較しない（HOT SAX については，ディスコード削除時はディスコードの再計算を行う）．

評価指標．1 スライドあたりの平均更新時間，最悪更新時間，および近似率（$= s^*.dist_{NN}/s_{out}.dist_{NN}$）を評価する．

5.2 評価結果

本節では，評価実験の結果を示す．l，w，および ϵ のデフォルトの値は 100，10,000，および 1.2 である．また，あるパラメータの影響を調べるとき，他のパラメータは固定する．

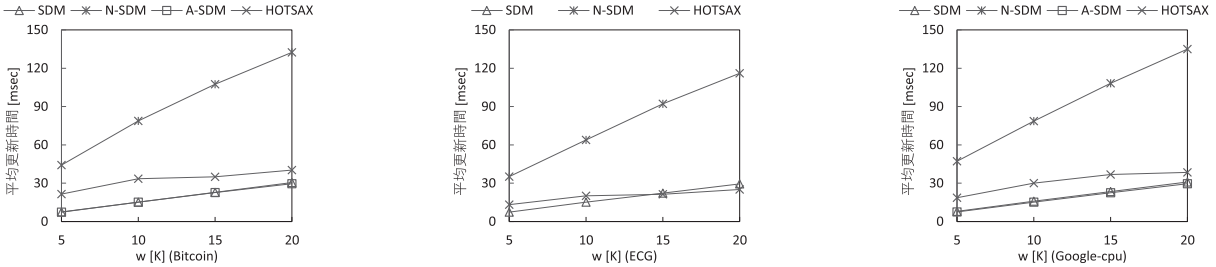
l の影響．図 4 に l を変化させたときの結果を示す．まず，図 4 (a)–(c) に示す平均更新時間について議論する．SDM の平均更新時間は，l の増加にともない，線形に増加する．これは，定理 2 より明らかである．次に，SDM および A-SDM は他のアルゴリズムより高速である．ウィンドウがスライドする際の Older-Nearest-Neighbor-Search(·) および Discord-Update の平均実行回数が小さいため，SDM および A-SDM は同様の結果となっている．また，インデックスベースのアルゴリズム HOT SAX より高速であるため，シーケンシャルスキャンが有効であることが分かる．さらに，SDM は N-SDM より大幅に高速である．これにより，4.2 節で説明したように，$\mathrm{NN}_{\mathrm{older}}$-タプルおよび $\mathrm{NN}_{\mathrm{younger}}$-タプルが効果的であることが分かる．

次に，図 4 (d)–(f) に示す最悪更新時間について議論する．SDM は，N-SDM と同等またはそれ以上の性能を示す．これは，各サブシーケンスが正確な最近傍を保持するからであり，HOT SAX よりも高速である．この結果は，静的な時系列データに対するアルゴリズムがストリーミング時系列データには適していないことを示している．また，A-SDM は SDM よりも最悪更新時間を短縮している（ECG での A-SDM の結果は SDM と同等の結果であったため省略している）．

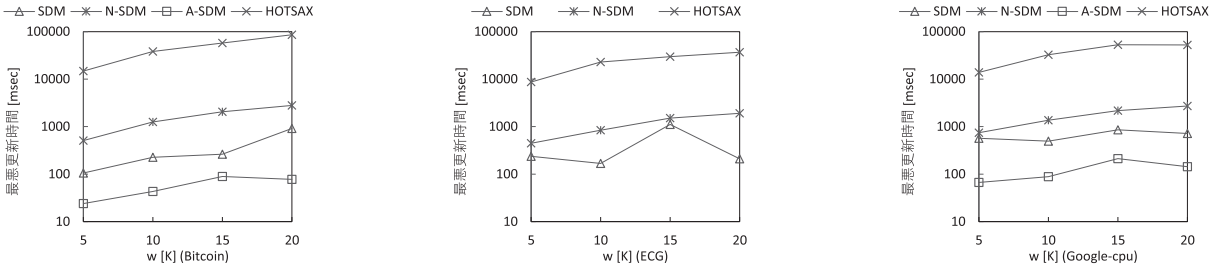
w の影響．図 5 に w を変化させたときの結果を示す．まず，図 5 (a)–(c) に示す平均更新時間について議論する．SDM および A-SDM は他のアルゴリズムより高速であり，w の増加にともない，線形に増加する．これは，定理 2 より明らかである．

次に，図 5 (d)–(f) に示す最悪更新時間について議論する．

[*1] http://api.bitcoincharts.com/v1/csv/

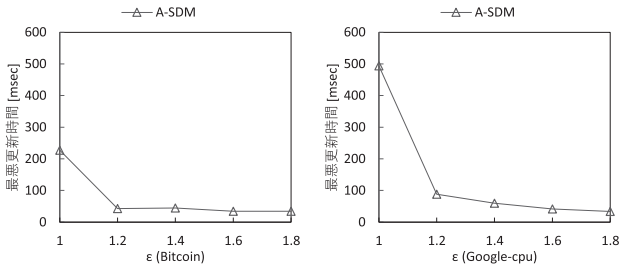


(a) 平均更新時間（Bitcoin） (b) 平均更新時間（ECG） (c) 平均更新時間（Google-cpu）

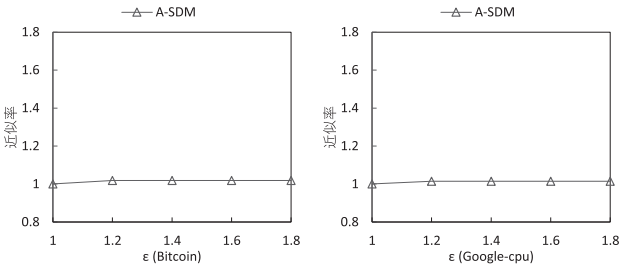


(d) 最悪更新時間（Bitcoin） (e) 最悪更新時間（ECG） (f) 最悪更新時間（Google-cpu）

図 5 w の影響
Fig. 5 Impact of w (window size).



(a) 最悪更新時間（Bitcoin） (b) 最悪更新時間（Google-cpu）



(c) 近似率（Bitcoin） (d) 近似率（Google-cpu）

図 6 ϵ の影響
Fig. 6 Impact of ϵ (approximation factor).

SDM の最悪更新時間が HOT SAX および N-SDM よりつねに高速であることが分かる．これは，SDM は $\mathrm{NN}_{\mathrm{older}}$-タプル，$\mathrm{NN}_{\mathrm{younger}}$-タプル，および NN-タプルを保持することで不必要な計算（最近傍探索）を削減しているからである．l の影響と同様に，A-SDM は最悪更新時間をさらに削減し，HOT SAX および N-SDM よりもはるかに高速である．

ϵ の影響． **図 6** に ϵ を変化させたときの結果を示す．理論的には，ϵ が増加するにつれて，更新時間は小さくなるが，得られるディスコードの精度は悪くなる．図 6 (a)–(b) より，ϵ が増加するにつれて，最悪更新時間は小さくなっており，直感的な結果となっている（平均更新時間は ϵ の影響を受けないことを確認したため省略している）．一方，図 6 (c)–(d) より，ϵ が増加しても近似率はほぼ 1（実際には 1.03 未満）であることが分かる．つまり，A-SDM は非常に精度の高いディスコードをモニタリングできる．

6. 結論

近年，多くのアプリケーションはストリーミング時系列データを生成しており，その時系列データの外れ値をリアルタイムにモニタリングすることで異常検知およびデータクリーニングに応用できる．そのため，本論文ではスライディングウィンドウ上でストリーミング時系列データのディスコードをモニタリングする問題に初めて取り組んだ．

この問題を解決するため，SDM（**S**treaming **D**iscord **M**onitoring）を提案した．SDM は，シーケンシャルスキャンによる最近傍探索を用いて，新たに生成されるサブシーケンスの最近傍を取得し，また，最近傍が変化するサブシーケンスを特定する．SDM は単純でありながら効率的であり，また，より効率的にディスコードをモニタリングできる近似アルゴリズムに拡張できる．実データを用いた評価実験により，SDM は効率的にディスコードをモニタリングでき，A-SDM が高精度のディスコードをモニタリングしながら，最悪更新時間を削減できることを確認した．

謝辞 本研究の一部は，基盤研究（A）（18H04095），基盤研究（B）（JP17KT0082），および若手研究（B）（JP16K16056）の研究助成によるものである．ここに記して謝意を表す．

参考文献

[1] Amagata, D. and Hara, T.: Mining top-k co-occurrence patterns across multiple streams, *TKDE*, Vol.29, No.10, pp.2249–2262 (2017).

[2] Begum, N. and Keogh, E.: Rare time series motif discovery from unbounded streams, *PVLDB*, Vol.8, No.2, pp.149–160 (2014).

[3] Bu, Y., Leung, T.-W., Fu, A.W.-C., Keogh, E., Pei, J. and Meshkin, S.: Wat: Finding top-k discords in time series database, *SDM*, pp.449–454 (2007).

[4] Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A. and Batista, G.: The UCR Time Series Classification Archive (2015), available from ⟨www.cs.ucr.edu/~eamonn/time_series_data/⟩.

[5] Chiu, B., Keogh, E. and Lonardi, S.: Probabilistic discovery of time series motifs, *KDD*, pp.493–498 (2003).

[6] Esling, P. and Agon, C.: Time-series data mining, *ACM Computing Surveys*, Vol.45, No.1, p.12 (2012).

[7] Gupta, M., Gao, J., Aggarwal, C.C. and Han, J.: Outlier detection for temporal data: A survey, *TKDE*, Vol.26, No.9, pp.2250–2267 (2014).

[8] Huang, T., Zhu, Y., Mao, Y., Li, X., Liu, M., Wu, Y., Ha, Y. and Dobbie, G.: Parallel discord discovery, *PAKDD*, pp.233–244 (2016).

[9] Kato, S., Amagata, D., Nishio, S. and Hara, T.: Monitoring range motif on streaming time-series, *DEXA*, pp.251–266 (2018).

[10] Kato, S., Amagata, D., Nishio, S. and Hara, T.: Discord monitoring for streaming time-series, *DEXA*, pp.79–94 (2019).

[11] Keogh, E., Lin, J. and Fu, A.: Hot sax: Efficiently finding the most unusual time series subsequence, *ICDM*, pp.226–233 (2005).

[12] Keogh, E., Lin, J., Lee, S.-H. and Van Herle, H.: Finding the most unusual time series subsequence: Algorithms and applications, *Knowledge and Information Systems*, Vol.11, No.1, pp.1–27 (2007).

[13] Lam, H.T., Pham, N.D. and Calders, T.: Online discovery of top-k similar motifs in time series data, *SDM*, pp.1004–1015 (2011).

[14] Li, Y., Zou, L., Zhang, H. and Zhao, D.: Computing longest increasing subsequences over sequential data streams, *PVLDB*, Vol.10, No.3, pp.181–192 (2016).

[15] Li, Y., Yiu, M.L., Gong, Z., et al.: Quick-motif: An efficient and scalable framework for exact motif discovery, *ICDE*, pp.579–590 (2015).

[16] Lin, J., Keogh, E., Lonardi, S. and Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms, *SIGMOD*, pp.2–11 (2003).

[17] Linardi, M., Zhu, Y., Palpanas, T. and Keogh, E.: Matrix profile X: Valmod-scalable discovery of variable-length motifs in data series, *SIGMOD*, pp.1053–1066 (2018).

[18] Reiss, C., Wilkes, J. and Hellerstein, J.L.: Google cluster-usage traces: Format+ schema, *Google Inc., White Paper*, pp.1–14 (2011).

[19] Sanchez, H. and Bustos, B.: Anomaly detection in streaming time series based on bounding boxes, *International Conference on Similarity Search and Applications*, pp.201–213 (2014).

[20] Teflioudi, C., Gemulla, R. and Mykytiuk, O.: Lemp: Fast retrieval of large entries in a matrix product, *SIGMOD*, pp.107–122 (2015).

[21] Wang, X., Lin, J., Patel, N. and Braun, M.: A self-learning and online algorithm for time series anomaly detection, with application in CPU manufacturing, *CIKM*, pp.1823–1832 (2016).

[22] Wei, L., Keogh, E. and Xi, X.: Saxually explicit images: Finding unusual shapes, *ICDM*, pp.711–720 (2006).

[23] Yankov, D., Keogh, E. and Rebbapragada, U.: Disk aware discord discovery: Finding unusual time series in terabyte sized datasets, *Knowledge and Information Systems*, Vol.17, No.2, pp.241–262 (2008).

[24] Yeh, C.-C.M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H.A., Silva, D.F., Mueen, A. and Keogh, E.: Matrix profile I: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets, *ICDM*, pp.1317–1322 (2016).

[25] Zhang, A., Song, S., Wang, J. and Yu, P.S.: Time series data cleaning: From anomaly detection to anomaly repairing, *PVLDB*, Vol.10, No.10, pp.1046–1057 (2017).

加藤 慎也 （学生会員）

2018 年大阪大学工学部電子情報工学科卒業．同大学大学院情報科学研究科博士前期課程在学中．時系列データにおけるデータ検索技術に関する研究に従事．

天方 大地 （正会員）

2012 年大阪大学工学部電子情報工学科卒業．2014 年同大学大学院情報科学研究科博士前期課程修了．2015 年同大学院情報科学研究科博士後期課程修了後，同年同大学院情報科学研究科マルチメディア工学専攻助教となり，現在に至る．情報科学博士．データベース，ネットワーク環境におけるデータ検索技術に関する研究に従事．IEEE，ACM，日本データベース学会各会員．

西尾 俊哉 （学生会員）

2016 年大阪大学工学部電子情報工学科卒業．2018 年同大学大学院情報科学研究科博士前期課程修了後，同年同大学院情報科学研究科博士後期課程在学中．データベース，ネットワーク環境におけるデータ検索技術に関する研究に従事．

原 隆浩 （正会員）

1995 年大阪大学工学部情報システム工学科卒業．1997 年同大学大学院工学研究科博士前期課程修了．同年同大学院工学研究科博士後期課程中退後，同大学院工学研究科助手，2004 年同大学院情報科学研究科准教授．2015 年より同大学院情報科学研究科教授となり，現在に至る．工学博士．2003 年本学会研究開発奨励賞受賞．2008 年，2009 年本学会論文賞，2015 年日本学術振興会賞受賞．ネットワーク環境におけるデータ管理技術に関する研究に従事．