ストリーミング時系列データの効率的なディスコードモニタリングアルゴリズム

加藤　慎也† 　天方　大地† 　西尾　俊哉† 　原　隆浩†

† 大阪大学大学院情報科学研究科　〒 565–0871 大阪府吹田市山田丘 1–5

E-mail: †{kato.shinya,amagata.daichi,nishio.syunya,hara}@ist.osaka-u.ac.jp

あらまし　近年，多くの IoT 機器はストリーミング時系列データを生成しており，それらを異常検知に応用することが考えられる．これを実現する一つの技術として，時系列データの中の各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も大きいサブシーケンスであるディスコードの発見が注目を集めている．本稿では，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．ウィンドウがスライドした際，新たにサブシーケンスが生成され，最も古いサブシーケンスが削除される．これらのサブシーケンスにより，ウィンドウ内のサブシーケンスの最近傍のサブシーケンスが変化し，ディスコードが変化する可能性がある．しかし，ウィンドウがスライドするたびにウィンドウ内の全てのサブシーケンスに対して最近傍探索を実行する手法は，ディスコードの更新に多大な時間がかかる．そのため，効率的にディスコードを更新するアルゴリズム SDM を提案する．4 つの実データを用いた実験により，SDM の有効性を確認する．

キーワード　ストリーミング時系列データ，ディスコードモニタリング

1　序　　論

近年，時系列データにおける外れ値の検出が注目を集めており，ディスコードの発見は外れ値を検出する最も重要な技術の一つである [5]．ある時系列データ t のディスコードとは，t を長さ l のサブシーケンスに分割したとき，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．例えば，図 1 は心電図のストリーミング時系列データを表しており，赤いサブシーケンスがディスコードである．多くの IoT 機器はストリーミング時系列データを生成するため [9]，本稿では，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．

アプリケーション例．心電図の時系列データをモニタリングする場合，このデータのディスコードをモニタリングすることで不整脈の検知を行うことができる．また，一般的に，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検出することで，リアルタイムなデータクリーニングが可能であり，時系列データマイニングのタスクの精度を高めることができる．

上記のようなアプリケーションにおいて過去の全ての値を考慮する場合，過去に外れ値があったとき，その外れ値が常にディスコードとなり，新たな外れ値を検出できない可能性がある．そのため，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 個の値のみを考慮し，それらのディスコードをモニタリングする．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，

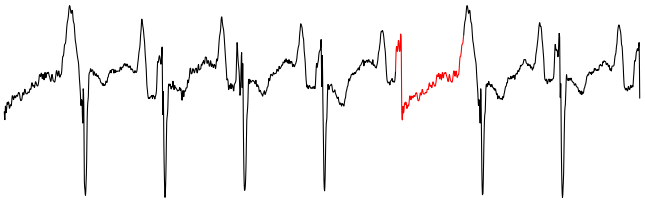


図 1: ディスコードの例

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．ディスコードをモニタリングする単純な手法は，ウィンドウがスライドした際，ウィンドウ内の全てのサブシーケンスに対して最近傍探索を実行するものである．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．

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 (Streaming Discord Monitoring) を提案する．SDM では，ウィンドウ内の各サブシーケンスに対して，最近傍および自身を最近傍とするサブシーケンスの情報を保持することで，ウィンドウのスライドにより最近傍が変化するサブシーケンスを効率的に発見する．

貢献．以下に本研究の貢献を示す．

- 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．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．

- ウィンドウがスライドした際にディスコードを効率的に更新するアルゴリズム SDM を提案する．
- 4 つの実データを用いた実験により，SDM の有効性を確認する．

本稿の構成． 2 章で本稿の問題を定義し，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 章で本稿のまとめと今後の課題について述べる．

2 予 備 知 識

2.1 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，$t = (t[1], t[2], ...)$ と表現する．まず，t の中に現れる通常のパターンと異なるパターンを発見するため，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 番目のデータを始点とするサブシーケンス 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，$s_p = (s_p[1], s_p[2], \cdots, s_p[l])$ である．次に，t の中で s_p の最近傍のサブシーケンスを計算するため，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．

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 $d(s_p, s_q)$ は式 (2) により定義される．

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

ここで，$\mu(s.)$ および $\sigma(s.)$ はそれぞれ $(s.[1], s.[2], ..., s.[l])$ 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 $O(l)$ 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．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 [2], [11]．

定義 3（トリビアルマッチ）． 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．

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

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 [1], [7]．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 [4], [6], [10] 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 $t = (t[i], t[i+1], ..., t[i+w-1])$ 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．本研究では，このような環境におい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をモニタリングする．つまり，ウィンドウサイズ w のウィンドウに含まれる全てのサブシーケンスの集合を S，サブシーケンス s の最近傍のサブシーケンスとの距離を $s.dist_{NN}$ 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，l，および w が与えられたとき，式 (4) で表されるディスコード s^* をモニタリングする．

$$s^* = \arg\max_{s \in S} s.dist_{NN} \tag{4}$$

3 関 連 研 究

本章では，ディスコードの発見およびストリーミング時系列データのモニタリングに関する既存研究について紹介する．

3.1 ディスコードの発見

文献 [5] では，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 は，まず SAX によってサブシーケンスを記号列に変換して管理する．そして，同じ記号列に変換されるサブシーケンス間の距離は小さくなる可能性が高いという性質を利用して，最近傍探索を高速化する．具体的には，現時点でのディスコードの最近傍距離が D だとすると，あるサブシーケンスに対して最近傍探索を行うとき，そのサブシーケンスの最近傍距離が D 以下になった時点で，そのサブシーケンスはディスコードになり得ないため，以降の計算を打ち切ることができる．文献 [14] では，ディスク上の時系列データに対するディスコード発見アルゴリズムを提案している．文献 [14] の提案アルゴリズムは，ディスコードの候補を発見するフェーズおよびディスコードの候補を絞り込むフェーズからなる．ディスコードの候補を発見するフェーズでは，あるサブシーケンスと他のサブシーケンスとの距離が閾値 r 以下である場合，ディスコードにはなり得ないため候補から除外する．ディスコードの候補を絞り込むフェーズでは，候補間の距離を計算し，閾値 r 以下の場合，候補から除外する．一方，r 以上の場合，暫定のディスコードの最近傍距離を更新する．文献 [3] では，ディスコード発見を並列化することで高速化するアルゴリズム PDD を提案している．PDD では，全てのサブシーケンスを記号列に変換し，同じ記号列に変換されたサブシーケンスをグループ化する．グループ内のサブシーケンスの数が少ない順に計算ノードに割り当てる．各計算ノードでは，グループ

内のサブシーケンスに関して最近傍探索を行い，最も最近傍距離が大きいサブシーケンスを求める．グループ内の計算をすべて終えると，他の計算ノードと通信を行い，暫定のディスコードの最近傍距離を更新する．グループ内のサブシーケンスの数が少ない順に計算していくため，早い段階で暫定のディスコードの最近傍距離が大きくなり，暫定のディスコードの最近傍距離による枝刈りを効率的に行うことができる．しかし，これらの研究は，静的な時系列データを対象としている．

3.2 ストリーミング時系列データのモニタリング

3.2.1 ディスコードのモニタリング

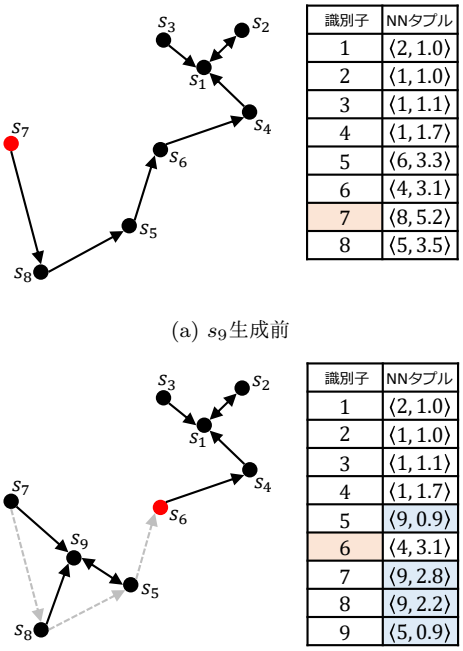
文献 [13] では，データの追加に対応したディスコードをモニタリングするアルゴリズムを提案している．文献 [13] の提案アルゴリズムでは，サブシーケンスを R 木で管理し，新たに生成されるサブシーケンスに対して R 木を用いた最近傍探索を行う．そして，新たに生成されるサブシーケンスの最近傍距離がディスコードの最近傍距離よりも大きい場合はディスコードを更新する．文献 [15] では，Matrix Profile と呼ばれるインデックスを用いてデータの追加に対応したディスコードをモニタリングするアルゴリズムを提案している．このインデックスは，全てのサブシーケンスに対して最近傍のサブシーケンスとの距離を保持する．新たなサブシーケンスが生成されると，そのサブシーケンスに対して最近傍探索を行い，Matrix Profile を更新する．そして，Matrix Profile が最大となるサブシーケンスがディスコードとなる．これらの研究はデータの削除には対応していない．

3.2.2 スライディングウィンドウ上でのストリーミング時系列データのモニタリング

文献 [10] では，カウントベースのスライディングウィンドウ上でペアモチーフをモニタリングする問題に取り組んでいる．ペアモチーフとは，時系列データの中で最も類似するサブシーケンスのペアである．文献 [10] の提案アルゴリズムでは，ペアモチーフを高速に更新するため，各サブシーケンスは近傍のサブシーケンスのリストおよび逆最近傍のサブシーケンスのリストを保持する．また，文献 [6] では，ペアモチーフをモニタリングするためにデータ構造を最適化したアルゴリズムを提案しており，文献 [10] のアルゴリズムより優れた性能を示している．文献 [4] では，カウントベースのスライディングウィンドウ上でレンジモチーフをモニタリングする問題に取り組んでいる．レンジモチーフとは，時系列データの中で最も多く現れるサブシーケンスである．文献 [4] の提案アルゴリズムでは，ウィンドウ内のサブシーケンスを Piecewize Aggregate Approximation で圧縮後，kd 木で管理している．kd 木を用いた範囲検索によりレンジモチーフが更新されるかどうかを高速に把握できる．

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



識別子	NNタプル
1	⟨2, 1.0⟩
2	⟨1, 1.0⟩
3	⟨1, 1.1⟩
4	⟨1, 1.7⟩
5	⟨6, 3.3⟩
6	⟨4, 3.1⟩
7	⟨8, 5.2⟩
8	⟨5, 3.5⟩

(a) s_9生成前

識別子	NNタプル
1	⟨2, 1.0⟩
2	⟨1, 1.0⟩
3	⟨1, 1.1⟩
4	⟨1, 1.7⟩
5	⟨9, 0.9⟩
6	⟨4, 3.1⟩
7	⟨9, 2.8⟩
8	⟨9, 2.2⟩
9	⟨5, 0.9⟩

(b) s_9生成後

図 2: サブシーケンス生成時の例

サブシーケンス s_e が削除される．s_n が生成されるとき，ウィンドウ内の各サブシーケンスの最近傍のサブシーケンスが変化し，ディスコードが変化する場合がある．また，s_e が削除されるときも同様にディスコードが変化する場合がある．さらに，s_e がディスコードの場合，つまり，ディスコードが削除されるとき，ウィンドウ内の全てのサブシーケンスに対して，それらの最近傍のサブシーケンスを計算する必要がある．

4.1 節において s_n が生成される際の処理を説明し，4.2 節において s_e が削除される際の処理を説明する．4.3 節において SDM の全体的なアルゴリズムを紹介し，SDM の計算量について述べる．最後に，4.4 節において，SDM の最悪更新時間を小さくするアルゴリズム I-SDM について説明する．

4.1 s_n の生成

新たなサブシーケンス s_n が生成されたとき，ウィンドウ内の各サブシーケンスの最近傍のサブシーケンスが変化するかどうかを効率的に把握するため，SDM では，ウィンドウ内の各サブシーケンスに対して最近傍の情報である NN (**N**earest **N**eighbor) タプルを管理する．

定義 4（NN タプル）．s_p の NN タプル $s_p.\langle id, dist\rangle_{NN}$ は，s_p の最近傍のサブシーケンスの識別子（id）と最近傍のサブシーケンスとの距離（$dist$）で構成される．

つまり，s_n が生成されたとき，$s_p \in S\backslash\{s_n\}$ に対して，$dist(s_n, s_p) < s_p.dist_{NN}$ ならば，s_p の最近傍は s_n に変化し，$s_p.\langle n, dist(s_n, s_p)\rangle_{NN}$ となる．

例 1. 図 2 に，サブシーケンスの生成による，各サブシーケンスの NN タプルの変化の例を示す．図 2 では，長さ 2 のサブシーケンスを 2 次元上の点として表現している．また，あるサ

識別子	NN_{older} タプル	$NN_{younger}$ タプル	NNタプル
1		$\langle 2, 1.0\rangle$	$\langle 2, 1.0\rangle$
2	$\langle 1, 1.0\rangle$	$\langle 4, 1.6\rangle$	$\langle 1, 1.0\rangle$
3	$\langle 1, 1.1\rangle$	$\langle 6, 2.9\rangle$	$\langle 1, 1.1\rangle$
4	$\langle 1, 1.7\rangle$	$\langle 6, 3.0\rangle$	$\langle 1, 1.7\rangle$
5		$\langle 6, 3.3\rangle$	$\langle 6, 3.3\rangle$
6	$\langle 4, 3.1\rangle$	$\langle 7, 5.3\rangle$	$\langle 4, 3.1\rangle$
7		$\langle 8, 5.2\rangle$	$\langle 8, 5.2\rangle$
8	$\langle 5, 3.5\rangle$		$\langle 5, 3.5\rangle$

(a) s_9生成前

識別子	NN_{older} タプル	$NN_{younger}$ タプル	NNタプル
1		$\langle 2, 1.0\rangle$	$\langle 2, 1.0\rangle$
2	$\langle 1, 1.0\rangle$	$\langle 4, 1.6\rangle$	$\langle 1, 1.0\rangle$
3	$\langle 1, 1.1\rangle$	$\langle 6, 2.9\rangle$	$\langle 1, 1.1\rangle$
4	$\langle 1, 1.7\rangle$	$\langle 6, 3.0\rangle$	$\langle 1, 1.7\rangle$
5		$\langle 9, 0.9\rangle$	$\langle 9, 0.9\rangle$
6	$\langle 4, 3.1\rangle$	$\langle 9, 3.2\rangle$	$\langle 4, 3.1\rangle$
7		$\langle 9, 2.8\rangle$	$\langle 9, 2.8\rangle$
8		$\langle 9, 2.2\rangle$	$\langle 9, 2.2\rangle$
9	$\langle 5, 0.9\rangle$		$\langle 5, 0.9\rangle$

(b) s_9生成後

図 3: NN_{older} タプルおよび $NN_{younger}$ タプルの使用例

ブシーケンス s_i から s_j への矢印は s_j が s_i の最近傍であることを表しており，例えば，s_6 の最近傍は s_4 である．時刻 t において，s_7 がディスコードである．時刻 $t+1$ に新たに s_9 が生成され，s_5，s_7，および s_8 の NN タプルが変化する．これにより，$s_6.dist_{NN}$ が最も大きい値となるため，ディスコードは s_6 に変化する．

s_n が生成されるとき，s_n とウィンドウ内の全サブシーケンスとの距離計算を行うことで，s_n の最近傍を計算する．そのため，ウィンドウ内のサブシーケンスは，自身より前に生成されたサブシーケンスに対する最近傍，および自身より後に生成されたサブシーケンスに対する最近傍を別々に保持したとすると，自身より後に生成されたサブシーケンスに対する最近傍を常に正確に保持することができる．そこで，各サブシーケンスに対して，自身より前に生成されたサブシーケンスに対する最近傍（NN_{older} タプル），および自身より後に生成されたサブシーケンスに対する最近傍（$NN_{younger}$ タプル）を管理する．

定義 5（NN_{older} タプル）．s_p の NN_{older} タプル $s_p.\langle id, dist\rangle_{NN_{older}}$ は，s_p より前に生成されたサブシーケンスに対する s_p の最近傍のサブシーケンスの識別子（id）と最近傍のサブシーケンスとの距離（$dist$）で構成される．

定義 6（$NN_{younger}$ タプル）．s_p の $NN_{younger}$ タプル $s_p.\langle id, dist\rangle_{NN_{younger}}$ は，s_p より後に生成されたサブシーケンスに対する s_p の最近傍のサブシーケンスの識別子（id）と最近傍のサブシーケンスとの距離（$dist$）で構成される．

定義 4，5，および 6 より以下の補題が成り立つ．

補題 1. $s_p.dist_{NN_{older}} < s_p.dist_{NN_{younger}} \Leftrightarrow s_p.\langle\cdot,\cdot\rangle_{NN} = s_p.\langle\cdot,\cdot\rangle_{NN_{older}}$

補題 2. $s_p.dist_{NN_{older}} > s_p.dist_{NN_{younger}} \Leftrightarrow s_p.\langle\cdot,\cdot\rangle_{NN} = s_p.\langle\cdot,\cdot\rangle_{NN_{younger}}$

ここで，あるサブシーケンス s_p に対して，$s_p.\langle\cdot,\cdot\rangle_{NN} = s_p.\langle\cdot,\cdot\rangle_{NN_{younger}}$ が成り立つ場合，$s_p.id_{NN_{younger}}$ より前に生成されたサブシーケンスが s_p の最近傍となることはないため，$s_p.\langle\cdot,\cdot\rangle_{NN_{older}}$ を保持する必要はない．

例 2. 図 3 に，サブシーケンスの生成による，各サブシーケンスの NN タプル，NN_{older} タプル，および $NN_{younger}$ タプルの変化の例を示す．時刻 t において，s_7 がディスコードであり，上で述べたように，$NN_{younger}$ タプルと NN タプルが等しい場合，NN_{older} タプルは保持しない．時刻 $t+1$ に新たに s_9 が生成され，s_9 に対して最近傍探索を行う．このとき，$dist(s_9, s_5) < s_5.dist_{NN_{younger}}$ であるため，$NN_{younger}$ タプルを更新する．同様に，s_6，s_7，および s_8 の $NN_{younger}$ タプルを更新する．さらに，$dist_{NN} > dist_{NN_{younger}}$ である場合，NN タプルを更新する．

4.2 s_e の削除

サブシーケンス s_e が削除されたとき，s_e を最近傍としていたサブシーケンスの最近傍が変化する．そこで，ウィンドウ内の各サブシーケンスに対して，自身を最近傍とするサブシーケンスの識別子のリスト RL (**R**everse Nearest Neighbor **L**ist) を保持すると，s_e が削除される際に，ウィンドウ内のサブシーケンスの最近傍が変化するかどうかを効率的に把握できる．つまり，あるサブシーケンス s_p の RL RL_p は以下の条件を満たす．

$$RL_p = \{q | s_q.id_{NN} = p\} \tag{5}$$

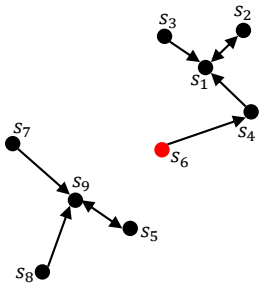
s_e が削除されるとき，s_e より前に生成されたサブシーケンスはすでに削除されているため，RL は自身より後に生成されたサブシーケンスの識別子のみを管理すればよい．また，s_e が削除される際に，RL_e に含まれるサブシーケンスの最近傍が変化するが，$s_p.dist_{NN_{younger}}$ がディスコードの最近傍距離より小さい場合，s_p の最近傍の計算をすることなく s_p がディスコードにならないことがわかる．

例 3. 図 4 に，サブシーケンスの削除による，各サブシーケンスの NN タプル，NN_{older} タプル，$NN_{younger}$ タプル，および RL の変化の例を示す．時刻 t において，s_6 がディスコードである．時刻 $t+1$ において，s_1 が削除されるとき，RL_1 に 2，3，および 4 が格納されているため，s_2，s_3，および s_4 の NN_{older} タプルおよび NN タプルが変化する．このとき，$s^*.dist_{NN} > s_2.dist_{NN_{younger}}$ であるため，s_2 に対して最近傍探索を行わずに，s_2 がディスコードにならないことがわかる．同様に，s_3 および s_4 もディスコードにならないことがわかる．

4.3 アルゴリズム

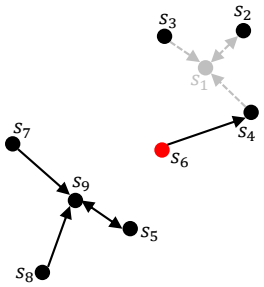
本節では SDM の詳細を紹介する．ウィンドウがスライドした際，まず，削除されるサブシーケンス s_e に対する処理を行い，その後，新たに生成されるサブシーケンス s_n に対する処理を行う．

s_e に対する処理． アルゴリズム 1 は，SDM の s_e に対する処理を行うアルゴリズムを示している．s_e が与えられたとき，ウィンドウ内のサブシーケンスの集合 S から s_e を削除する（1 行）．s_e がディスコード s^* である場合，Discord-Update を実行し，s^* を取得する（2–3 行）．Discord-Update の詳細は後述する．次に，RL_e に対する処理を行う（4–9 行）．s_e の削除により，s_p の NN_{older} タプル $s_p.\langle\cdot,\cdot\rangle_{NN_{older}}$ および s_p の NN タプル $s_p.\langle\cdot,\cdot\rangle_{NN}$ が変化するため，初期化を行う．次に，



識別子	NN_{older} タプル	$NN_{younger}$ タプル	NNタプル	RL
1		$\langle 2, 1.0\rangle$	$\langle 2, 1.0\rangle$	2, 3, 4
2	$\langle 1, 1.0\rangle$	$\langle 4, 1.6\rangle$	$\langle 1, 1.0\rangle$	
3	$\langle 1, 1.1\rangle$	$\langle 6, 2.9\rangle$	$\langle 1, 1.1\rangle$	
4	$\langle 1, 1.7\rangle$	$\langle 6, 3.0\rangle$	$\langle 1, 1.7\rangle$	6
5		$\langle 9, 0.9\rangle$	$\langle 9, 0.9\rangle$	9
6	$\langle 4, 3.1\rangle$	$\langle 9, 3.2\rangle$	$\langle 4, 3.1\rangle$	
7		$\langle 9, 2.8\rangle$	$\langle 9, 2.8\rangle$	
8		$\langle 9, 2.2\rangle$	$\langle 9, 2.2\rangle$	
9	$\langle 5, 0.9\rangle$		$\langle 5, 0.9\rangle$	

(a) s_1削除前



識別子	NN_{older} タプル	$NN_{younger}$ タプル	NNタプル	RL
2		$\langle 4, 1.6\rangle$		
3		$\langle 6, 2.9\rangle$		
4		$\langle 6, 3.0\rangle$		6
5		$\langle 9, 0.9\rangle$	$\langle 9, 0.9\rangle$	9
6	$\langle 4, 3.1\rangle$	$\langle 9, 3.2\rangle$	$\langle 4, 3.1\rangle$	
7		$\langle 9, 2.8\rangle$	$\langle 9, 2.8\rangle$	
8		$\langle 9, 2.2\rangle$	$\langle 9, 2.2\rangle$	
9	$\langle 5, 0.9\rangle$		$\langle 5, 0.9\rangle$	

(b) s_1削除後

図 4: サブシーケンス削除時の例

Algorithm 1: SDM (expiration case)

Input: s_e: the expired subsequence
Output: s^*: the discord

```
1  S ← S\{s_e}
2  if s_e = s* then
3      s* ← Discord-Update
4  for ∀p ∈ RL_e do
5      p.⟨·,·⟩_{NN_older} ← ∅, p.⟨·,·⟩_{NN} ← ∅
6      if s*.dist_NN < p.dist_{NN_younger} then
7          NNSearch(p)
8          if s*.dist_NN < p.dist_NN then
9              s* ← p
```

$s_p.dist_{NN_{younger}}$ が $s^*.dist_{NN}$ より小さい場合，s_p はディスコードになり得ないため，処理を終了する．$s_p.dist_{NN_{younger}}$ が $s^*.dist_{NN}$ より大きい場合，s_p はディスコードになり得るため，s_p に対して最近傍探索 NNSearch(s_p) を実行する．NNSearch の詳細は後述する．s_p の最近傍距離が $s^*.dist_{NN}$ より大きい場合，s_p を s^* とする．

s_n に対する処理． アルゴリズム 2 は，SDM の s_n に対する処理を行うアルゴリズムを示している．s_n が与えられたとき，ウィンドウ内のサブシーケンスの集合 S に s_n を追加し，$s_n.\langle\cdot,\cdot\rangle_{NN}$,$s_n.\langle\cdot,\cdot\rangle_{NN_{older}}$,$s_n.\langle\cdot,\cdot\rangle_{NN_{younger}}$ を初期化する（1–2 行）．次に，s_n に対して NNSearch(s_n) を実行する（3 行）．$s_n dist_{NN}$ が $s^*.dist_{NN}$ より大きい場合は，s_n を s^* とする（4–5 行）．

Algorithm 2: SDM (insertion case)

Input: s_n: the new subsequence
Output: s^*: the discord

```
1  S ← S ∪ {s_n}
2  s_n.⟨·,·⟩_{NN} ← ∅, s_n.⟨·,·⟩_{NN_older} ← ∅,
     s_n.⟨·,·⟩_{NN_younger} ← ∅
3  NNSearch(s_n)
4  if s_n.dist_NN > s*.dist_NN then
5      s* ← s_n
```

Algorithm 3: NNSearch(s_p)

Input: s_p: a subsequence

```
1  for ∀s_q ∈ S such that p > q do
2      d ← dist(s_p, s_q)
3      if s_p.dist_{NN_older} > d then
4          s_p.⟨·,·⟩_{NN_older} ← ⟨q, d⟩
5      if s_q.dist_{NN_younger} > d then
6          s_q.⟨·,·⟩_{NN_younger} ← ⟨p, d⟩
7          if s_q.dist_NN > s_q.dist_{NN_younger} then
8              if s_q.⟨·,·⟩_NN = s_q.⟨·,·⟩_{NN_older} then
9                  RL_{s_q.id_NN} ← RL_{s_q.id_NN}\{q}
10             s_q.⟨·,·⟩_NN ← s_q.⟨·,·⟩_{NN_younger}
11             if s_q = s* then
12                 s* ← Discord-Update
13 if s_p.dist_{NN_older} < s_p.dist_{NN_younger} then
14     s_p.⟨·,·⟩_NN ← s_p.⟨·,·⟩_{NN_older}
15     RL_{s_p.id_NN} ← RL_{s_p.id_NN} ∪ {p}
16 else
17     s_p.⟨·,·⟩_NN ← s_p.⟨·,·⟩_{NN_younger}
```

最近傍探索． アルゴリズム 3 は，最近傍探索を行うアルゴリズムを示している．s_p と $p > q$ を満たす s_q との距離 d を計算し，$s_p.\langle\cdot,\cdot\rangle_{NN_{older}}$ を更新する（2–4 行）．$s_q.dist_{NN_{younger}}$ より d が小さい場合，$s_q.\langle\cdot,\cdot\rangle_{NN_{younger}}$ を更新する（5–6 行）．さらに，$s_q.dist_{NN}$ より $s_q.dist_{NN_{younger}}$ が大きい場合，$s_q.\langle\cdot,\cdot\rangle_{NN}$ を更新し，$s_q.\langle\cdot,\cdot\rangle_{NN} = s_q.\langle\cdot,\cdot\rangle_{NN_{older}}$ 場合，つまり $RL_{s_q.id_{NN}}$ に q 含まれている場合，$RL_{s_q.id_{NN}}$ から q を削除する（7–10 行）．また，s_q が s^* である場合，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（11–12 行）．$s_p.dist_{NN_{older}}$ が $s_p.dist_{NN_{younger}}$ より小さい場合，$s_p.\langle\cdot,\cdot\rangle_{NN_{older}}$ を $s_p.\langle\cdot,\cdot\rangle_{NN}$ に代入し，$RL_{s_p.id_{NN}}$ に p を追加する（13–15 行）．$s_p.dist_{NN_{older}}$ が $s_p.dist_{NN_{younger}}$ より大きい場合，$s_p.\langle\cdot,\cdot\rangle_{NN_{younger}}$ を $s_p.\langle\cdot,\cdot\rangle_{NN}$ に代入する（16–17 行）．

ディスコードの更新． アルゴリズム 4 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．まず，暫定のディスコードの NN タプル $s^*_{temp}.\langle\cdot,\cdot\rangle_{NN}$ を初期化する（1 行）．ウィンドウ内のサブシーケンスをスキャンし，$s_p.\langle\cdot,\cdot\rangle_{NN} = \varnothing$ であるサブシーケンスを $dist_{NN_{younger}}$ を降順に格納するヒープ H に追加する

Algorithm 4: Discord-Update

Output: s^*_{temp}: a temporal discord

1 $s^*_{temp}.\langle\cdot,\cdot\rangle_{NN} \leftarrow \langle -1, 0\rangle$
2 **for** $\forall s_p \in S$ **do**
3 **if** $s_p.\langle\cdot,\cdot\rangle_{NN} = \varnothing$ **then**
4 Up-Max-Heap(H, s_p)
5 **else**
6 **if** $s^*_{temp}.dist_{NN} < s_p.dist_{NN}$ **then**
7 $s^*_{temp} \leftarrow s_p$
8 **while** $H \neq \varnothing$ **do**
9 $s_p \leftarrow$ Down-Max-Heap(H)
10 **if** $s^*_{temp}.dist_{NN} > s_p.dist_{NN_{younger}}$ **then**
11 **break**
12 **else**
13 NNSearch(s_p)
14 **if** $s^*_{temp}.dist_{NN} < s_p.dist_{NN}$ **then**
15 $s^*_{temp} \leftarrow s_p$
16 **return** s^*_{temp}

(3–4 行)．$s_p.\langle\cdot,\cdot\rangle_{NN} = \varnothing$ であるサブシーケンス s_p の最近傍距離が $s^*_{temp}.dist_{NN}$ よりも大きい場合，s_p を s^*_{temp} とする（5–7 行）．次に，ヒープ H に関する処理を行う（8–15 行）．H の先頭から 1 つサブシーケンス s_p を取り出し，$s_p.dist_{NN_{younger}}$ が $s^*_{temp}.dist_{NN}$ より小さい場合，s_p および H 内のサブシーケンスはディスコードにならないため，処理を終了する（9–11 行）．$s_p.dist_{NN_{younger}}$ が $s^*_{temp}.dist_{NN}$ より大きい場合，s_p はディスコードになる可能性があるため，NNSearch(s_p) を実行する（13 行）．$s_p.dist_{NN}$ が $s^*_{temp}.dist_{NN}$ より大きい場合，s_p を s^*_{temp} とする（14–15 行）．最後に，s^*_{temp} を返却する（16 行）．

時間計算量．サブシーケンス間の距離計算には $O(l)$ 時間かかり，ウィンドウ内には $w - l + 1$ 個のサブシーケンスがある．そのため，NNSearch には $O(wl)$ 時間かかる．Discord-Update における H の構築では，最大 $w - l + 1$ 回ヒープへの挿入が行われるため $O(w \log w)$ 時間かかる．また，H 構築後の処理では，最大 $w - l + 1$ 回 NNSearch が実行されるため，$O(w^2 l)$ 時間かかる．そのため，Discord-Update には $O(w^2 l)$ 時間かかる．アルゴリズム 1 において，Discord-Update が実行されるとき，$O(w^2 l)$ 時間かかり，また，$|RL_e|$ 回 NNSearch 実行される．$|RL_e|$ は最大 $w - l$ であるため，アルゴリズム 1 の時間計算量は $O(w^2 l)$ となる．アルゴリズム 2 は，NNSearch が一度実行されるため，時間計算量は $O(wl)$ となる．以上より，SDM の時間計算量は $O(w^2 l)$ となる．

空間計算量．ウィンドウ内の各サブシーケンスは，NN タプル，$\mathrm{NN}_{\mathrm{older}}$ タプル，$\mathrm{NN}_{\mathrm{younger}}$ タプル，および RL を保持する．各サブシーケンスの保持する NN タプル，$\mathrm{NN}_{\mathrm{older}}$ タプル，および $\mathrm{NN}_{\mathrm{younger}}$ タプルの空間計算量は $O(w)$ であり，各サブシーケンスの RL の要素数の合計は高々 $w - l$ 個であるため，SDM の空間計算量は $O(w)$ となる．

表 1: パラメータ設定

パラメータ	値
ディスコード長，l	50, **100**, 150, 200
ウィンドウサイズ，w [$\times 10^3$]	5, **10**, 15, 20

4.4 I-SDM

SDM において，NN タプルを保持しないサブシーケンスの数が多い場合，Discord-Update 実行時の最近傍探索の実行回数が多くなることで，ディスコードの最悪更新時間が大きくなる可能性がある．そこで，本節では，SDM におけるディスコードの最悪更新時間を小さくするアルゴリズム I-SDM を提案する．

I-SDM では，サブシーケンスの削除により最近傍が変化し，NN タプルが空となった場合，そのサブシーケンスをリストに追加する．そして，サブシーケンス削除時に一度も最近傍探索が実行されない場合，そのリストから一つサブシーケンスを取り出し，そのサブシーケンスに対して最近傍探索を実行する．このようにして，Discord-Update 実行時の最近傍探索の実行回数を削減できる．

5 性能評価

本章では，SDM および I-SDM の性能評価のために行った実験の結果を紹介する．SDM および I-SDM の性能を評価するため，以下のアルゴリズムとの比較を行った．

N-SDM. N-SDM では，ウィンド内の各サブシーケンスは NN タプルおよび RL を保持する．サブシーケンス生成時は SDM と同様の処理を実行し，サブシーケンス削除時は RL によって最近傍が変化するサブシーケンスを特定し，そのサブシーケンスに対して最近傍探索を実行する．また，ディスコードが変化する可能性がある場合，ウィンドウ内のサブシーケンスの NN タプルをスキャンし，最近傍距離が最大となるサブシーケンスをディスコードとする．

5.1 実験環境

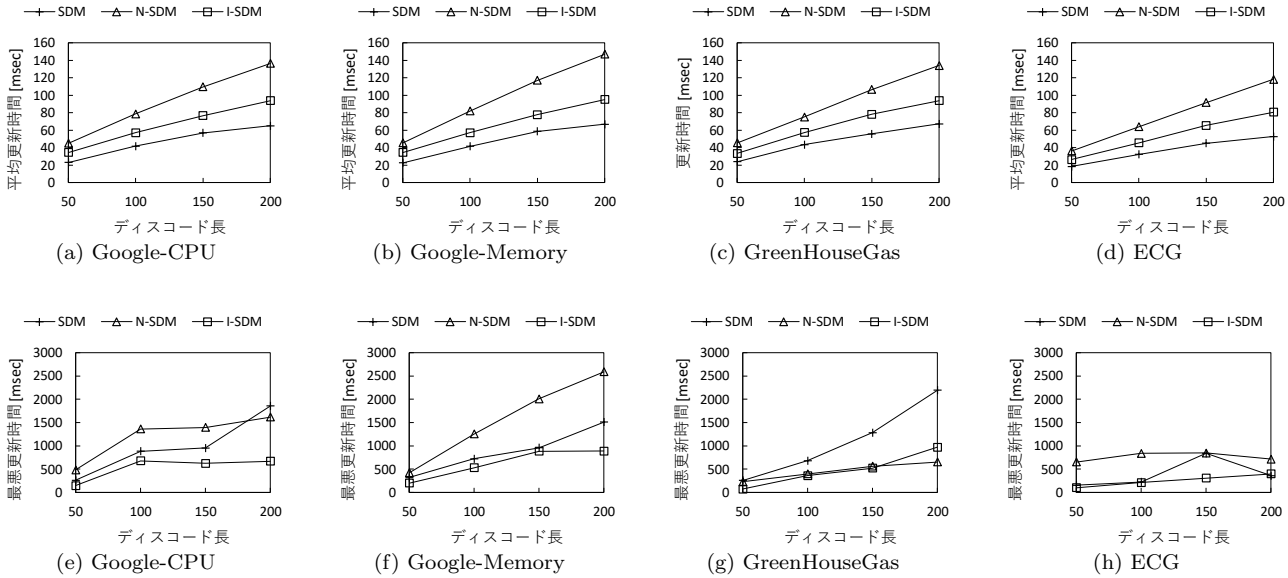
本実験は，Windows 10 Pro for Workstations，3.00GHz Intel Xeon Gold，および 512GB RAM を搭載した計算機で行い，全てのアルゴリズムを C++で実装した．

データセット．以下の 4 つの実データを用いた．

- Google-CPU [12]: Google のデータセンタの CPU 使用率の時系列データ（長さ 133,902）
- Google-Memory [12]: Google のデータセンタのメモリ使用率の時系列データ（長さ 133,269）
- GreenHouseGas [8]: カリフォルニア州の温室効果ガスの排出量の時系列データ（長さ 100,062）
- ECG[1]: 心電図の時系列データ（長さ 100,000）

パラメータ．本実験で用いたパラメータを表 1 に示す．太字で表されている値はデフォルトの値である．また，あるパラメー

1 : http://timeseriesclassification.com/index.php



(a) Google-CPU (b) Google-Memory (c) GreenHouseGas (d) ECG

(e) Google-CPU (f) Google-Memory (g) GreenHouseGas (h) ECG

図 5: l の影響（(a)–(d)：平均更新時間，(e)–(h：最悪更新時間）

タの影響を調べるとき，他のパラメータは固定する．

評価指標．1 スライド当たりの平均更新時間，および最悪更新時間を評価する．

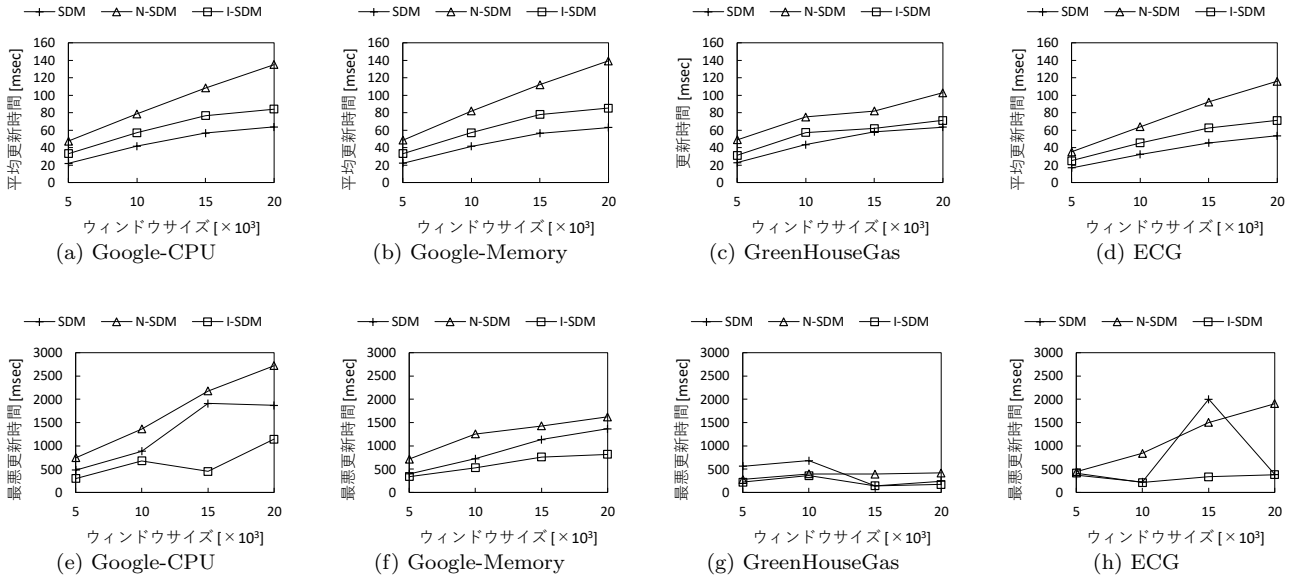
5.2 評価結果

l の影響．図 5 に l を変化させたときの結果を示す．全てのアルゴリズムにおける平均更新時間は，l の増加に伴い，線形に増加する．これは，l の増加に伴い，サブシーケンス間の距離の計算時間が大きくなり，最近傍探索にかかる時間が増加するからである．また，平均更新時間は常に N-SDM が最大であり，次に I-SDM が大きく，SDM が最小となっている．これは，サブシーケンス削除時，N-SDM では RL に含まれる全てのサブシーケンスに対して最近傍探索を実行するが，SDM および I-SDM ではディスコードになり得るサブシーケンスに対してのみ最近傍探索を実行するからである．また，サブシーケンス削除時，SDM では一度も最近傍探索が実行されない場合があるが，I-SDM ではそのような場合，NN タプルを保持しないサブシーケンスに対して最近傍探索を実行するため，I-SDM の平均更新時間は SDM より大きくなる．

最悪更新時間は，l の増加に伴って増加する傾向がある．これは，l の増加に伴い，サブシーケンス間の距離の計算時間が大きくなるからである．一方で，最悪更新時間が l の増加に伴って減少する場合もあるが，これは，l によって RL のサイズが偏ったり，ディスコードが変化する場合に必要な最近傍探索の回数が変化するからである．N-SDM では，削除されるサブシーケンスの RL が大きいとき，最近傍探索の実行回数が増加し，最悪更新時間が悪くなる．SDM では，削除されるサブシーケンスの RL が大きい場合でもディスコードになり得ないサブシーケンスに対しては最近傍探索を実行しないが，一方で，ディスコードが変化するとき，NN タプルを保持していない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る．GreenHouseGas において，SDM の最悪更新時間が大きいのは上記の理由によるものである．I-SDM では，ディスコードが変化する際の最近傍探索の実行回数が SDM より小さくなっており，いずれのデータセットにおいても，他のアルゴリズムより最悪更新時間が小さくなっている．

w の影響．図 6 に w を変化させたときの結果を示す．全てのアルゴリズムにおける平均更新時間は，w の増加に伴い，線形に増加する．これは，w の増加に伴い，ウィンドウ内のサブシーケンスの数が増加し，最近傍探索にかかる時間が増加するからである．また，l を変化させたときと同様の理由で，平均更新時間は常に N-SDM が最大であり，次に I-SDM が大きく，SDM が最小となっている．

最悪更新時間は，w の増加に伴って増加する傾向がある．これは，w の増加に伴い，ウィンドウ内のサブシーケンスの数が増加し，最近傍探索にかかる時間が増加するからである．また，l を変化させたときと同様の理由で，w の増加に伴って最悪更新時間が減少する場合もある．I-SDM では，ディスコードが変化する際の最近傍探索の実行回数が SDM より小さくなっており，いずれのデータセットにおいても，他のアルゴリズムより最悪更新時間が小さくなっている．Google-CPU および Google-Memory において，最悪更新時間は N-SDM が最大であり，次に SDM が大きくなっている．これは，ディスコードが変化する際の最近傍探索の影響よりも，サブシーケンス削除時の最近傍探索の影響が大きいからである．一方，GreenHouseGas および ECG においては，N-SDM と SDM の最悪更新時間が逆転している場合がある．これは，w によってサブシーケンス削除時の最近傍探索の実行回数や，ディスコードが変化する場合に必要な最近傍探索の回数が変化するからである．



(a) Google-CPU (b) Google-Memory (c) GreenHouseGas (d) ECG

(e) Google-CPU (f) Google-Memory (g) GreenHouseGas (h) ECG

図 6: w の影響（(a)–(d)：平均更新時間，(e)–(h)：最悪更新時間）

6 結　　論

近年注目されている IoT 機器では，ストリーミング時系列データを生成するため，時系列データをリアルタイムに解析することがより重要になっている．本稿では，ディスコード（時系列データの中で他と最も異なるサブシーケンス）をモニタリングする問題に初めて取り組んだ．効率的にディスコードをモニタリングするため，SDM を提案した．SDM は，最近傍のサブシーケンスおよび自身を最近傍とするサブシーケンスのリストを保持することにより，効率的にディスコードをモニタリングできる．4 つの実データを用いた実験により，SDM の有効性を確認した．

謝辞．本研究の一部は，基盤研究 (A)(JP18H04095)，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．

文　　献

[1] Y. Chen, M.A. Nascimento, B.C. Ooi, and A.K. Tung, “Spade: On shape-based pattern detection in streaming time series,” ICDE, pp.786–795, 2007.

[2] B. Chiu, E. Keogh, and S. Lonardi, “Probabilistic discovery of time series motifs,” KDD, pp.493–498, 2003.

[3] T. Huang, Y. Zhu, Y. Mao, X. Li, M. Liu, Y. Wu, Y. Ha, and G. Dobbie, “Parallel discord discovery,” PAKDD, pp.233–244, 2016.

[4] S. Kato, D. Amagata, S. Nishio, and T. Hara, “Monitoring range motif on streaming time-series,” DEXA, pp.251–266, 2018.

[5] E. Keogh, J. Lin, and A. Fu, “HOT SAX: Efficiently finding the most unusual time series subsequence,” ICDM, pp.226–233, 2005.

[6] H.T. Lam, N.D. Pham, and T. Calders, “Online discovery of top-k similar motifs in time series data,” SDM, pp.1004–1015, 2011.

[7] Y. Li, L. Zou, H. Zhang, and D. Zhao, “Computing longest increasing subsequences over sequential data streams,” PVLDB, vol.10, no.3, pp.181–192, 2016.

[8] D. Lucas, C.Y. Kwok, P. Cameron-Smith, H. Graven, D. Bergmann, T. Guilderson, R. Weiss, and R. Keeling, “Designing optimal greenhouse gas observing networks that consider performance and cost,” Geoscientific Instrumentation, Methods and Data Systems, vol.4, no.1, p.121, 2015.

[9] M. Moshtaghi, C. Leckie, and J.C. Bezdek, “Online clustering of multivariate time-series,” SDM, pp.360–368, 2016.

[10] A. Mueen and E. Keogh, “Online discovery and maintenance of time series motifs,” KDD, pp.1089–1098, 2010.

[11] P. Patel, E. Keogh, J. Lin, and S. Lonardi, “Mining motifs in massive time series databases,” ICDM, pp.370–377, 2002.

[12] C. Reiss, J. Wilkes, and J.L. Hellerstein, “Google cluster-usage traces: format+ schema,” Google Inc., White Paper, pp.1–14, 2011.

[13] H. Sanchez and B. Bustos, “Anomaly detection in streaming time series based on bounding boxes,” SISAP, pp.201–213, 2014.

[14] D. Yankov, E. Keogh, and U. Rebbapragada, “Disk aware discord discovery: Finding unusual time series in terabyte sized datasets,” KIS, vol.17, no.2, pp.241–262, 2008.

[15] C.C.M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H.A. Dau, D.F. Silva, A. Mueen, and E. Keogh, “Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets,” ICDM, pp.1317–1322, 2016.