動的に進化するデータベースを対象にした継続的な類似検索

野口　大樹†　　古賀　久志†

† 電気通信大学情報理工学研究科　〒 182–8585 東京都調布市調布ヶ丘 1–5–1
E-mail: †{noguchi.d,koga}@sd.is.uec.ac.jp

あらまし　近年，テクノロジーの進化によってストリームデータを解析する重要性が増している．本研究では，ストリームデータを対象とする類似検索を取り扱う．とくに，データストリームのスライディングウィンドウ内の直近の要素群を動的に変化する集合とみなすモデルを考える．従来研究では，クエリが動的に変化する集合 (Continuous Evolving Query, CEQ 問題) で，静的なデータベースから類似集合を発見する top-k 類似検索が取り扱われている．そして，データベース内の各集合に対して現在時刻の類似度の上限値を計算し，それを用いて top-k に入る可能性がある集合に対してのみ類似度を計算する枝刈りベースの厳密解アルゴリズムが開発された．

これに対して，本研究ではクエリが静的な集合で，多数のデータストリーム群から作られる動的に変化する集合群 (Continuous Evolving Database, CED 問題) を対象とする top-k 類似検索を考える．我々は，この問題に対して類似度の上限値を利用した厳密解アルゴリズムを提案する．提案アルゴリズムは CEQ 問題に対する厳密解アルゴリズムを CED 問題に拡張しているが，上限値の計算方法を厳密化する 2 つの工夫により，単純に拡張したバージョンよりも類似検索を高速化する．

キーワード　類似検索, ストリームデータ, アルゴリズム, 集合

1　はじめに

近年，ソーシャルネットワークや IoT (Internet of Things) の発展に伴い，ストリームデータ解析の重要性が高まっている．その中でストリームデータを対象とする類似検索は，リアルタイムでの情報推薦や異常検出の基盤となる技術として注目されている．ストリームデータでは時間経過に伴って新しいデータが追加される．これまで，ストリームデータを対象とした類似検索では，1 つのデータストリームをデータベースと見なし，データベース内でデータが動的に減ったり，増えたりする状況で静的なクエリと類似したデータを探し続ける問題が多く取り扱われている．[1] [2] [3]

一方で最近，データストリームをデータの集合と捉え，集合間で類似検索を実施する新しい問題設定が取り扱われ始めている．例えば，Efstathiades [4] らは twitter において類似ユーザを発見するために，各ユーザのツイート集合間で類似検索を行った (ただし，ツイート集合は静的なスナップショットを使用)．また，Xu ら [5] はデータストリームのスライディングウィンドウ内の要素群をクエリ集合として，集合のデータベースからクエリと最も類似した上位 k 個のデータ (集合) を探すことを目的とする「Continuous Similarity Search for Evolving Query」を提唱した．この問題では，時間経過によってデータストリームに新しい要素が来ると，スライディングウィンドウがスライドしてクエリが変化し (Evolving Query)，最も類似した上位 k 個の集合を更新する必要がある (Continuous Similarity Search)．この問題は，時間によるユーザの嗜好の変化に適応的に情報推薦を行う状況をモデル化したもので，Evolving Query がユーザの嗜好の変化，検索結果がユーザの嗜好に合わせて推薦されるオブジェクトに相当する．この問題に対して，Xu ら [5] は過去に計算した類似度の値から現在時刻で上位 k 位に入る可能性がある集合を決定し，それらに対してのみ類似度を計算する枝刈りベースの厳密解法を考案した．以降，本稿では Continuous Similarity Search for Evolving Query 問題を CEQ 問題と呼ぶ．

CEQ 問題ではクエリがデータストリームであり，データベースは複数の静的な集合で構成される．これに対して，本研究では，逆にクエリが静的な集合で，データベースに複数のデータストリームが登録されている状況で，クエリと類似した集合を検索する問題を考える．CEQ 問題と同様に各データストリームのスライディングウィンドウ内の要素群を集合と見なすと，本問題では要素追加によりスライディングウィンドウが移動することでデータベース内の集合が変質する (Evolving Database)．そして本問題では，データベースの変化に適応して最も類似した上位 k 個のデータを更新する (Continuous Similarity Search) 必要がある．以下では，Continuous Similarity Search for Evolving Database 問題を CED 問題と記述する．CED 問題の応用例としては，ウェブ広告をどのユーザに提示するかという情報推薦が挙げられる．ウェブ広告をカテゴリを示すキーワード集合で表し，個々のユーザをウェブページ閲覧履歴というデータストリームで特徴表現する状況を考える．この時，データストリームのスライディングウィンドウ内の履歴はユーザの直近の嗜好を表す．ここでさらに，ウェブページもカテゴリも表すキーワード集合と紐付ければ，スライディングウィンドウ内の履歴は時間と共に変化するキーワード集合となる．そして，ウェブ広告をクエリとして，CED 問題を解くことにより，ウェブ広告と嗜好性が適合する上位 k 人のユーザがを発見できる．

本研究では，CED 問題に対する枝刈りベースの厳密解法を提案する．提案アルゴリズムは CEQ 問題に対する厳密解法を CED 問題に拡張しているが，類似度の上限値を厳密に求めるための以下の 2 つの工夫により，単純に拡張したバージョンよりも類似検索を高速化することに成功した．

（1） CEQ 問題に対する厳密解法では類似度が再計算されるまで上限値を更新しないことで上限値計算のオーバーヘッドを減らしていた．我々の提案手法ではあえて上限値を毎時刻更新する．このアプローチは上限値計算のオーバーヘッドを増やすが，より厳密な上限値を求めることで類似度計算回数を減らしてトータルで計算コストを削減する．

（2） スライディングウィンドウは，将来入って来る要素は不明であるが，将来離脱する要素は分かっているという特徴を持つ．この将来離脱する要素の情報を利用してより厳密な類似度の上限値を計算する．

2 CEQ 問題 (Continuous Similarity Search over Evolving Query)

本節では，Xu らによって提唱された既存の CEQ 問題の定義を述べた後，CEQ 問題に対する厳密解法を 2 つ紹介する．$\psi = \{x_1, x_2, \cdots, x_{|\psi|}\}$ をアルファベットとする．

CEQ 問題ではデータストリームは 1 つだけ登場する．これを X とする．X には，毎時刻新しい要素が 1 つ到着する．時刻 t に到着する要素を $e_t \in \psi$ とする．時刻 t のクエリ Q_t は X の直近 w 個の要素 $\{e_{t-w+1}, e_{t-w+2}, \cdots, e_{t-1}, e_t\}$ である．この直近 w 個の要素をスライディングウィンドウ内のデータと呼ぶ．一方，データベース D は, n 個のアルファベットを要素とする集合 $\{S_1, S_2, \cdots, S_n\}$ で構成される．D 内の集合は静的であり時間によって変化しない．

CEQ 問題では，毎時刻，クエリと最も類似した上位 k 個の集合 (top-k) をデータベース D から検索する．時刻 t が変化するとクエリ Q_t も変化し，検索結果も変わるので検索結果を更新することが要求される．なお，集合間類似度には jaccard 係数

$$\mathrm{sim}(S, Q_t) = \frac{|S \cap Q_t|}{|S \cup Q_t|} = \frac{|S \cap Q_t|}{|S| + |Q_t| - |S \cap Q_t|} \qquad (1)$$

を用いる．

本節では，CEQ 問題に対する既存アルゴリズムを 2 つ紹介する．1 つ目は Xu ら [5] による枝刈りベースのアルゴリズムであり，2 つ目は山崎ら [6] による転置インデックスベースのアルゴリズムである．

2.1 枝刈による厳密解法

Xu らは枝刈りベースの厳密解法 GP(general pruning-based method) [5] を考案した．GP では，データベース D 内の各集合 S に対して類似度の上限値を軽量な手法で求めておき，それが現在時刻における top-k 類似度の下限値を下回る場合に，S に対する類似度計算を省略して実行速度を高速化する．

2.1.1 類似度の上限値

時刻 t にクエリ Q_t と集合 S_i 間で類似度 $sim(S_i, Q_t) = \frac{|S_i \cap Q_t|}{|S_i \cup Q_t|}$ を計算したとしよう．GP では，S_i に対して類似度の再計算が必要となる時刻 $t + \mathrm{min\_step}_i$ を推定し，時刻 t から時刻 $t + \mathrm{min\_step}_i$ の期間で有効な類似度の上限値を算出する．$\mathrm{min\_step}_i$ は類似度の再計算が必要となるまでの時間の推定値である．時間が $\mathrm{min\_step}_i$ 経過すると，クエリ Q_t の要素が $\mathrm{min\_step}_i$ 個入れ替わることから, 期間 $[t, t + \mathrm{min\_step}_i]$ での類似度の上限値は式 (3) となる．

$$\frac{|S_i \cap Q_t| + \mathrm{min\_step}_i}{|S_i \cup Q_t| - \mathrm{min\_step}_i} \qquad (2)$$

$$= \frac{(|S_i| + |Q_t| + \mathrm{min\_step}_i)\mathrm{sim}(S_i, Q_t) + \mathrm{min\_step}_i}{|S_i| + |Q_t| - \mathrm{min\_step}_i \cdot \mathrm{sim}(S_i, Q_t) - \mathrm{min\_step}_i} \qquad (3)$$

式 (2) から式 (3) への式変形は，まず $|S_i \cup Q_t| = |S_i| + |Q_t| - |S_i \cap Q_t|$ という関係を用いて式 (2) から $|S_i \cup Q_t|$ の項を除去し，次に残った $|S_i \cap Q_t|$ の項を Jaccard 係数の定義 (1) を用いて $\mathrm{sim}(S_i, Q_t)$ で表すことでできる．式 (3) を計算するには，$\mathrm{min\_step}_i$ の値が必要となる．GP では $\mathrm{min\_step}_i$ を，式 (3) が時刻 t の top-k 類似度 $\mathrm{score}(topk^t)$ を越えるのに必要な最低限の経過時間と定める．つまり，$\mathrm{min\_step}_i$ は $\mathrm{score}(topk^t) < \frac{|S_i \cap Q_t| + \mathrm{min\_step}_i}{|S_i \cup Q_t| - \mathrm{min\_step}_i}$ を満たす最小の整数である．

GP は，期間 $[t, t + \mathrm{min\_step}_i]$ の間は上限値を更新しない．

2.1.2 top-k 類似集合の算出

時刻 T において，クエリ Q_T と類似した上位 k 個の集合を求める手順は以下のようになる．データベース D に登録された集合を，$t_i + \mathrm{min\_step}_i = T$ を満足する集合群 $R = \{S_i | t_i + \mathrm{min\_step}_i = T\}$ とそれ以外 $D \backslash R$ に分割する．ここで t_i は，S_i に対して時刻 T 以前で最後に類似度を計算した時刻である．

まず，R に含まれる集合は類似度が時刻 t_i での top-k より大きい可能性があるので類似度を再計算する．R の全集合に対して類似度を再計算した後, R の中で top-k 類似集合を求める．この時, k 番目に類似した集合の類似度は，D の中での top-k 類似集合に対する類似度の下限値 lb として使える．$D \backslash R$ 内の集合 S_i に対しては, S_i の上限値が lb を下回る場合に類似度計算を省略する．なお，$D \backslash R$ 内の集合に対して類似度を計算した結果，類似度が lb を上回った場合は，lb を増加させる．

2.2 転置インデックスを使用する方法

山崎らによる厳密解法 [6] では，時刻が t から $t+1$ に進んだとき, 類似度が変化する可能性のない集合を転置インデックスを用いて高速に特定し類似度計算を省略する．時刻が進んだときに，クエリのスライディングウインドウに入る要素を IN，スライディングウインドウからでる要素を OUT とする．この時，データベース内の集合 S が IN と OUT のどちらも含まなければ，$\mathrm{sim}(S, Q_t) = \mathrm{sim}(S, Q_{t+1})$ となり類似度は変わらないので，類似度計算を省略して構わない．逆に言えば，IN と OUT のどちらかを含む集合に対してのみ類似度計算をすればよく, 山崎らの手法ではこの条件を満足する集合を転置インデックスで高速に見つける．さらに類似度を計算する場合，$\mathrm{sim}(S, Q_{t+1})$ は $\mathrm{sim}(S, Q_t)$ の値を $O(1)$ の計算時間で更新することで非常に高速に求める．この手法は枝刈ベースの GP より 10 倍以上，

高速に動作することが示されている，

3 CED 問題 (Continuous Similarity Search over Evolving Database)

本節では，本研究で取り扱う新しい問題である CED 問題を定式化する．CEQ 問題と同様に，$\psi = \{x_1, x_2, \cdots, x_{|\psi|}\}$ をアルファベットとする．

CED 問題では，クエリ Q はアルファベットの集合である．Q は時間によって変化せず静的である．一方，データベース D には n 個のデータストリーム群 $\{X_1, X_2, \cdots, X_n\}$ が登録されている．各データストリーム X_i には幅 w のスライディングウィンドウが設定され，スライディングウィンドウには X_i に到着した直近 w 個の要素が含まれる．CED 問題では時刻 t のスライディングウィンドウ内の要素群を集合 S_i^t と見なす．その結果，時刻 t のデータベース D_t は要素数が等しい n 個の集合 $\{S_1^t, S_2^t, \cdots, S_n^t\}$ で構成される．毎時刻，クエリ Q と最も類似した上位 k 個の集合 (top-k) をデータベース D_t から検索することが CED 問題の目的である．時刻 t が変化するとデータベース D_t も変化するので，検索結果を更新することが要求される．なお，集合間類似度には jaccard 係数

$$\mathrm{sim}(S_t, Q) = \frac{|S_t \cap Q|}{|S_t \cup Q|}$$

を使用する．本稿では，問題を簡単化するため，データストリームにデータが到着するレートはデータストリームによらず一定であり，毎時刻，新しいデータが 1 つ到着することを仮定する．

次節では CED 問題に対する枝刈りベースの厳密解法を提案する．この手法では転置インデックスを使用しない．従来の CEQ 問題では，データベースに登録された集合は静的で不変であった．このため，一度データベースの全集合を転置インデックスに登録すれば，それ以降，転置インデックスの更新は不要であり，CEQ 問題は転置インデックスとの相性がよい．一方，本研究で取り扱う CED 問題では毎時刻，全集合に対して要素の到着と離脱が発生するため，転置インデックスへの集合の挿入と削除が大量に発生する．とくに，リスト構造の転置インデックスからの削除はコストが大きい．このように，転置インデックスをメンテナンスするオーバーヘッドが膨大であることを考慮し，本研究では枝刈りベースの厳密解法を研究対象とした．

4 提 案 手 法

本節では提案手法となる CED 問題に対する厳密解法について述べる．提案手法は CEQ 問題に対する厳密解法を CED 問題に拡張しているが，類似度の上限値を厳密に算出するように工夫することで，単純に拡張したバージョンよりも類似検索を高速化する．本節では，4.1 節で CEQ 問題に対する厳密解法が CED 問題に拡張できることを延べ，4.2 節，4.3 節で類似度の上限値を厳密に算出するための工夫点を解説する．

4.1 CEQ 問題に対する厳密解法の CED 問題への適用

CEQ 問題と CED 問題との差異は, クエリとデータのどちらが静的で, どちらが動的かという点である．CEQ 問題に対する厳密解法において，クエリを動的な Q_t から静的な Q に，データベース側を静的な $\{S_1, S_2, \cdots, S_n\}$ から，動的な $\{S_1^t, S_2^t, \cdots, S_n^t\}$ に置換すると，CED 問題に対する厳密解法を構築できる．

実際，時刻 t にクエリ Q と集合 S_i^t 間で類似度 $\mathrm{sim}(S_i^t, Q) = \frac{|S_i^t \cap Q|}{|S_i^t \cup Q|}$ が計算されたとすると，時刻 $t + \mathrm{min\_step}_i$ における類似度の上限値は，データストリーム X_i のスライディングウィンドウの要素が毎時刻 1 つ入れ替わるという条件下で

$$\frac{|S_i^t \cap Q| + \mathrm{min\_step}_i}{|S_i^t \cup Q| - \mathrm{min\_step}_i} \tag{4}$$

となる．これはまさに CEQ 問題に対する類似度上限値の式 (2) において，$Q_t \to Q$, $S_i \to S_i^t$ と置換したものである．同様に $\mathrm{min\_step}_i$ も $\mathrm{score}(topk^t) < \frac{|S_i^t \cap Q| + \mathrm{min\_step}_i}{|S_i^t \cup Q| - \mathrm{min\_step}_i}$ を満たす最小の整数と定義できる．また，2.1.2 節で述べた top-k 類似集合の算出手順は修正無しでそのまま適用可能である．

上記の手順で，CEQ 問題に対する厳密解法を単純に CED 問題に適用した厳密解法を GPD(General Pruning algorithm for evolving Database) と名付ける．

4.2 上限値の毎時刻更新

4.1 節で述べた GPD では，CEQ 問題に対する GP と同様に，集合 S_i に対して期間 $[t, t + \mathrm{min\_step}_i]$ における類似度の上限値を $\frac{|S_i^t \cap Q| + \mathrm{min\_step}_i}{|S_i^t \cup Q| - \mathrm{min\_step}_i}$ に固定する．この方法では時刻 t から $t + \mathrm{min\_step}_i$ までの間，記憶された変数を参照するだけで類似度の上限値を取り出すことが可能であり，上限値算出にかかるオーバーヘッドを抑制できるという利点がある．

しかしこのやり方は，時刻 $t+\alpha$ $(1 \leqq \alpha \leqq \mathrm{min\_step}_i)$ の類似度上限値がタイトでないという欠点を持つ．つまり，上限値が必要以上に高くなり, 類似度計算回数の増加につながる危険性がある．時刻 $t+\alpha$ には S_i^t の要素が α 個入れ替わっていることを考慮すると，類似度上限値を $\frac{|S_i^t \cap Q| + \alpha}{|S_i^t \cup Q| - \alpha} < \frac{|S_i^t \cap Q| + \mathrm{min\_step}_i}{|S_i^t \cup Q| - \mathrm{min\_step}_i}$ とよりタイトにできる．

そこで提案手法では類似度上限値を毎時刻更新し，時刻 $t+\alpha$ の類似度上限値を

$$\frac{|S_i^t \cap Q| + \alpha}{|S_i^t \cup Q| - \alpha} \tag{5}$$

に設定するように工夫した．時刻 t に類似度 $sim(S_i^t, Q)$ を類似度を求める過程で積集合の大きさ $|S_i^t \cap Q|$ は分かっているため, その値を記録すれば式 (5) は $\frac{|S_i^t \cap Q| + \alpha}{|S_i^t| + |Q| - |S_i^t \cap Q| - \alpha}$ として $O(1)$ の時間で計算できる．これは Jaccard 係数の計算時間 $O(|S_i^t| + |Q|) = O(w + |Q|)$ よりも十分小さい．本工夫は，上限値の計算回数は増やす一方で類似計算回数を減らし，類似検索全体の実行時間を削減することを狙うものである．

さらに, 時刻 T に top-k 類似集合を算出する際には，集合群 $R = \{$ 時刻 $T-1$ に top-k に入った k 個の集合 $\}$ とし，初期下限値 lb を R の中で最も類似度が低い集合の類似度とした．これにより, 集合 S_i に対する min_step_i の計算を削減した．

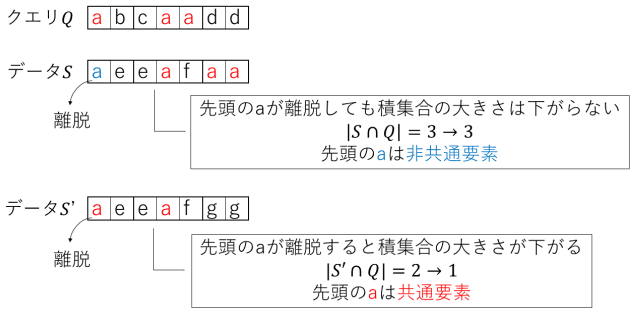

図 1　共 通 要 素

4.3　離脱要素に注目した上限値の高精度化

GP [5] 及び GPD における上限値は, スライディングウインドウに挿入される要素と離脱する要素がどちらも任意であるという悲観的な仮定の下で計算を行っている. しかし, 現実には, 将来離脱する要素は既にスライディングウインドウ内にあるため既知である．提案手法では，離脱要素の情報を利用して上限値の改善を試みる.

4.3.1　共 通 要 素

時刻 t にクエリ Q と集合 S_i^t 間で類似度 $\mathrm{sim}(S_i^t, Q) = \frac{|S_i^t \cap Q|}{|S_i^t \cup Q|}$ が計算された状況を考える．集合 S_i^t の w 個の要素を到着時刻順に $\{a_1, a_2, \cdots, a_w\}$ とする．$\forall 1 \leqq j \leqq w$ に対して，a_j はアルファベットであり，$a_j \in \psi$ を満たす．また，a_j は時刻 $t+j$ にスライディングウィンドウから離脱する.

[定義 1]（共通要素） 時刻 $t+j$ に a_J が離脱することにより，クエリ Q との積集合の大きさが $|S_i^{t+j-1} \cap Q|$ より 1 減る時，a_j を S_i と Q との共通要素と呼ぶ. (図 1)

この定義より，時刻 $t+j$ にスライディングウィンドウに入ってくる要素 a_{w+j} が何であっても，

$$|S_i^{t+j-1} \cap Q| \leqq |S_i^{t+j} \cap Q|. \tag{6}$$

が成立する．つまり，時刻 $t+j$ に積集合の大きさが前時刻から増加することはない.

さて，a_j のアルファベットラベルを α とすると，a_j が共通要素になるかどうかは時刻 $t+j-1$ にクエリ Q と S_i^{t+j-1} のどちらが α をより多く含むかにより決まる．クエリ Q が α を含む個数を $\#\alpha_Q$, S_i^{t+j-1} が α を含む個数を $\#\alpha_{S_i}$ とすると，a_j が共通要素になるかどうかの条件は次のように記述できる.

- $\#\alpha_{S_i} \leqq \#\alpha_Q$ であれば，a_j は共通要素である.
- $\#\alpha_{S_i} > \#\alpha_Q$ であれば，a_j は共通要素でない.

ここまで説明したように，a_j が共通要素であるかどうかは時刻 $t+j$ に定まる点に注意されたい．一方，時刻 t の時点で将来共通要素になる可能性がある S_i^t の要素を共通要素候補と定義する.

[定義 2]（共通要素候補） 集合 S_i^t の要素 a_j が，時刻 $t+j$ に共通要素になる可能性がある時，a_j のことを共通要素候補と呼ぶ.

S_i^t 内で a_j より後方に α が $\#\alpha_Q$ 個以上存在すれば，S_i^{t+j-1} は α を $\#\alpha_Q+1$ 個以上含むので，a_j は共通要素になる可能性はない．逆に a_j より後方に α が $\#\alpha_Q$ 個未満しか存在しない場合，期間 $[t+1, t+j-1]$ に α がデータストリーム X_i に一度も到着しなければ，S_i^{t+j-1} を高々$\#\alpha_Q$ 個しか含まないので a_j は共通要素となる可能性がある．以上の性質を補題 1 にまとめる.

[補題 1]　S_i^t 内で a_j より後ろに α が $\#\alpha_Q$ 個以上存在するならば，a_j は共通要素候補ではない．逆に S_i^t 内で a_j より後ろに α が $\#\alpha_Q$ 個未満しか存在しないのならば，a_j は共通要素候補である.

補題 1 から S_i^t の各要素が共通要素候補かどうかは Q と S_i^t のみから決定できる．クエリ Q と集合 S_i^t 間で Jaccard 係数 $\mathrm{sim}(S_i^t, Q)$ を計算するのと同時に，共通要素候補かどうかは効率よく判定可能である.

共通要素候補の検出:

ラベル α の a_j が共通要素候補となる条件は，自分より候補に α が $\#\alpha_Q$ 個未満しか存在しないことである．すなわち，S_i^t 内で最後方の $\min\{\#\alpha_{S_i^t}, \#\alpha_Q\}$ 個の α が共通要素候補となる．この条件を満たす α はクエリ Q と集合 S_i^t 間で Jaccard 係数 $\mathrm{sim}(S_i^t, Q)$ を計算する時に容易に発見できる.

提案手法では，Jaccard 係数を計算する直前に S_i^t をアルファベット順に並び替え，順序ありリスト L で表現する．この並び替え時に，同じアルファベット同士では到着時刻が遅いものほど優先 (前に配置する) すると，L 内では同じアルファベットの要素群は到着時刻の降順で連続することになる．順序ありリストで表現されたクエリ Q と L 間で Jaccard 係数を求める際には，リスト L 内で前方にある要素が積集合のメンバとしてカウントされる．L 内の α の個数が $\#\alpha_Q+1$ 以上ならば，前方の $\#\alpha_Q$ 個が積集合に寄与するが，これらは S_i^t 内で最高方の $\#\alpha_Q$ 個の α に対応し，共通要素候補そのものである．L 内の α の個数が $\#\alpha_Q$ 以下ならば，そのすべてが共通要素候補でかつ積集合に寄与するため問題はない.

このように，同じアルファベット同士では到着時刻が遅いものを優先して S_i^t に対する順序付きリストを生成し，Q との積集合メンバになった要素の到着時刻を調べると，S_i^t の任意の u 個の要素 $\{a_1, a_2, \cdots, a_u\}$ 内に含まれる共通要素候補の個数が分かる.

4.3.2　積集合サイズの上限値

この節では，S_i^t 内の共通要素候補の数を用いて，時刻 t 以降の S_i と Q との積集合サイズの上限値をより精密に求めることができることを述べる．具体的には以下の定理 1 が成り立つ.

[定理 1]　u を w 以下の任意の自然数とする．S_i^t の先頭の u 個の要素 $\{a_1, a_2, \cdots, a_u\}$ 内に共通要素候補が n 個あったとする．この時，時刻 $t+u$ における積集合 $S_i^{t+u} \cap Q$ のサイズは $|S_i^t \cap Q| + (u-n)$ を超えない．すなわち，

$$|S_i^{t+u} \cap Q| \leqq |S_i^t \cap Q| + (u-n).$$

4.3.3　上限値の高精度化

時刻 t に S_i^t の先頭の u 個の要素 $\{a_1, a_2, \cdots, a_u\}$ 内に共通要素候補が n 個あった場合，定理 1 より積集合のサイズは

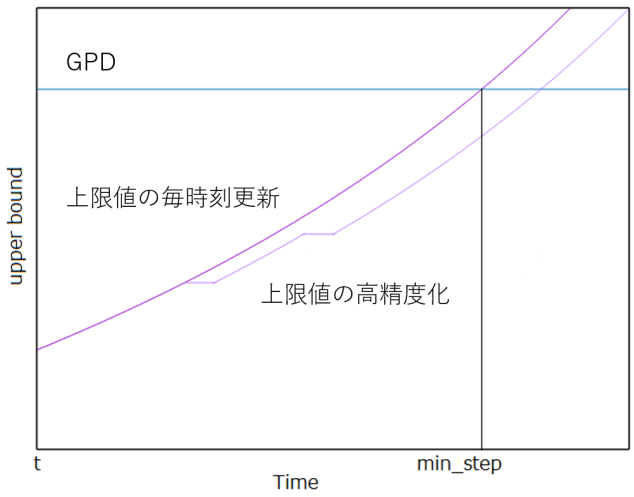

図 2　上限値の動き

高々$|S_i^t \cap Q| + (u-n)$ である．よって，時刻 $t+u$ の類似度 $\mathrm{sim}(S_i^{t+u}, Q)$ の上限値は

$$\frac{|S_i^t \cap Q| + (u-n)}{|S_i^t \cup Q| - (u-n)} \tag{7}$$

となる．

高精度化した上限値が時間経過に伴ってどう変化するかを図 2 に例示する．時々，上限値が増加しない時刻が発生するが，これは共通要素候補が共通要素として離脱した時刻と対応する．

4.3.4　適応的な高精度化

離脱要素に着目して類似度上限値を高精度化する工夫はクエリ Q と S_i^t 間で共通要素候補数が多いほど効果が大きい．しかし逆に共通要素候補が少ない場合は効果が小さく，共通要素候補であるかどうかを判定するオーバーヘッドに見合わない危険性がある．例えば，共通要素候補が 1 つもない場合は，明らかに共通要素判定のオーバーヘッドに見合わない．

そこで，共通要素候補数が多い場合だけ共通要素候補の判定を試みる．しかし，共通要素候補が多いかどうかは共通要素候補の判定をしないと不明なので，1 回前の類似度計算した時の共通要素候補数 $|Q \cap S_i^p|$ を現在の共通要素候補数 $|Q \cap S_i^t|$ の近似値として代用する．より具体的には $|Q \cap S_i^p| >$ 閾値β を満足する時のみ共通要素候補の判定を実施し，類似度上限値を高精度化する．以上の提案手法のアルゴリズムを Algorithm 1 に示す．

閾値 β を適切に定めるために，本項では高精度化の効力をモデル化する．

共通要素候補判定と類似度計算は同じタイミングで行われるので, その両方を行った際のコストを $jc2$, 類似度係数計算のみのコストを jc, 上限値の高精度化によって類似度計算を行う間隔は長くなるため, 高精度化を行った場合の間隔を st', 行わなかった場合の間隔を st, とすると高精度化が効力を持つための条件式は次のようになる．

$$\frac{jc2}{jc} < \frac{st'}{st}. \tag{8}$$

一方で，st' と st の間には次の関係式が成り立つ．

Algorithm 1 提案手法

Require: 時刻 $T-1$ の Top-k リスト, 時刻 T のデータベース D_t
Ensure: 時刻 T の Top-k リスト
  for each $S_i \in$ 時刻 $T-1$ の top-k リスト **do**
    if 時刻 $T-1$ の類似度 $>$ 閾値 β **then**
      類似度計算+共通要素候補判定
    else
      類似度計算のみ
    end if
    Top-k リストに登録
  end for
  類似度下限値 lb の設定
  for each $S_i \notin$ 時刻 $T-1$ の top-k リスト **do**
    上限値の更新
    if 上限値 $> lb$ **then**
      if 時刻 $T-1$ の類似度 $>$ 閾値 β **then**
        類似度計算+共通要素候補判定
      else
        類似度計算のみ
      end if
      if $\mathrm{sim}(S_i, Q) > lb$ **then**
        Top-k リストに登録して lb 更新
      end if
    end if
  end for

$$\frac{|S_i^t \cap Q|}{w} \cdot st' = st' - st \tag{9}$$

この式より，$st' = \frac{st \cdot w}{w - |S_i^t \cap Q|}$ となり，これを式 (8) に代入すると

$$\frac{jc2}{jc} < \frac{w}{w - |S_i^t \cap Q|}. \tag{10}$$

となる．式 (10) を $|S_i^t \cap Q|$ に関して解くと

$$w(1 - \frac{jc}{jc2}) < |S_i^t \cap Q| \tag{11}$$

以上より, 共通要素候補数の閾値 $\beta = w(1 - \frac{jc}{jc2})$.
jc と $jc2$ の値は計算機上での実測値を使用する．

5　実　　験

この節では, 提案手法の性能を実験的に評価する．実験のプラットフォームは, メモリ 16GB, Ubuntu18.04, Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz である．実験は人工データを用いた実験と実データを用いた実験の 2 種類を行った．

5.1　人工データによる実験

人工データの生成方法を説明する．最初に IBM Quest data generator [7] を使って,

- アルファベット種類数 $|\psi|$
- 集合の大きさ w

を指定して p 個の集合で構成される集合プールを生成する．クエリはこの集合プールからランダムに 1 つ選択することにより生成する．また, データベース側の 1 データストリームは, 集合

プール内の集合をランダムに複数個取り出して連結することで作成する．この方法で n 個のデータストリームを生成し，データベースに登録された n 個の集合とする．p をデータパターン数と呼ぶ．

5.1.1 jc と $jc2$ の実測値

4.3.4 項で述べた適応的に上限値の高精度化を行う手法での，共通要素候補数の閾値 β を決定するため，jc, $jc2$ の比を実測した．本実験ではスライディングウインドウサイズ $w = 100$, データパターン数 $p = 1000$, データベースの集合数 $n = 10000$, アルファベット種類数 $|\psi| = 200$ という条件の下で jaccard 係数だけ計算するプログラムと jaccard 係数と共通要素候補判定の両方を実行するプログラムを作成し，処理時間を実測した．

実行結果を図 3 に示した．実行結果の近似直線の傾きから jaccard 係数計算時間と jaccard 係数計算時間＋共通要素候補の判定時間の比は $jc : jc2 = 1169.3 : 1200.1$ と推測できる．よって以降はこの値を用いて β を算出する．

5.1.2 性 能 比 較

各時刻 t において，クエリ Q と類似度が高い k 個の集合を検索する実験を行った．時刻を $t = 1 \sim 1000$ まで進め，1000 回の top-k 検索にかかる合計の処理時間と類似度計算回数を測定した．実験は，スライディングウインドウサイズ $w = 100$, データパターン数 $p = 1000$, データベースの集合数 $n = 10000$ 求める top-k リストのサイズ $k = 10$ という条件の下でアルファベット種類数 $|\psi|$. を変化させて行った．

まず，提案手法を全ての集合と毎時刻類似度計算する単純手法と従来手法 GPD と比較した．単純手法，GPD，提案手法の実行時間を図 4 に示した．また，類似度計算回数を図 6 に示した．3 つの手法のなかで，提案手法が最も高速であったため，提案手法が効果的であることが確認できた．提案手法の実行時間は，$|\psi| = 200$ で最も遅く，$|\psi| = 1000$ 以降では速度が上がっていた．これにより，$|\psi|$ の値によって大きな速度の変化が出ていない単純手法，GPD との差が $|\psi| = 200$ では詰まり，$|\psi| = 1000$ 以降では広がるという結果になった．提案手法の実行時間がこのように変化した理由は上限値の毎時刻更新の効果が $|\psi| = 200$ の時に弱まるためである．

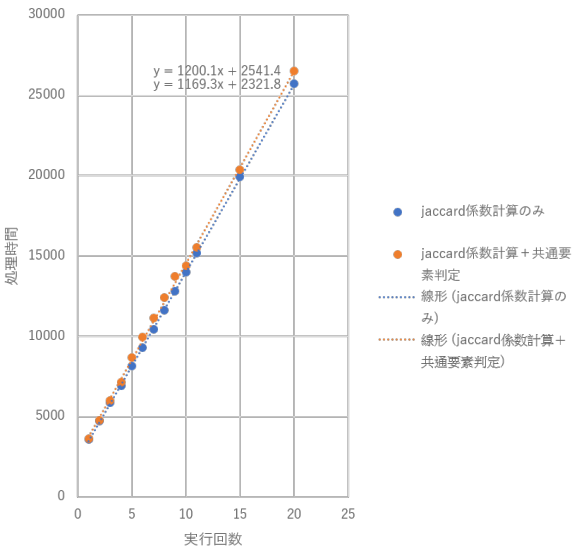

図 3　Jaccard 係数及び共通要素判定の実行時間

上限値の毎時刻更新の効果が $|\psi| = 200$ の時に弱まった理由は，top-k 類似度とデータベースの平均類似度が近い値になったためである．以下でこの説明をする．データパターン $p = 1000$ の集合プールから，クエリと $n = 10000$ 個のデータストリームを生成した場合，各時刻において，クエリと同じデータパターンの一部をスライディングウィンドウ内に含むデータストリームは，約 10〜20 個存在する．クエリと同じデータパターンを含む集合は類似度が明らかに高くなる．このデータベースから上位 10 個のデータを取り出すため，top-k の下限値とデータベースの平均類似度の距離は大きなものになる．この性質により，毎時刻更新が効果的になる．しかし，$|\psi| = 200$ の場合にはデータパターン同士の類似度が高いということが起こる．そのため，クエリと同じデータパターンを持たない集合の類似度も高くなり，top-k の下限値とデータベースの平均類似度の差が小さい．よって，毎時刻更新の効果が薄くなり低速化してしまう．上記の毎時刻更新の効果の減少は類似度計算回数のグラフから確認できる．以上を裏付けるために第 10 位の集合とクエリの積集合サイズと上位 20 位までのデータを除いたデータベースの平均積集合サイズを表 1 に示した．これを見ると $|\psi| = 200$ では 21 位以下の平均積集合サイズが多く第 10 位との差が小さいが，$|\psi| = 1000$ 以降では差が大きい．よって，top-k の下限値とデータベースの平均類似度の差が毎時刻更新の効果に影響を及ぼしていると考えられる．

次に，適応的な上限値の高精度化の効力を測るため

- 上限値の毎時刻更新を行う手法
- 毎時刻更新+すべてのデータに対して上限値の高精度化を行う手法
- 高精度化を行うデータを限定する提案手法

を比較した．実行結果を図 5 に示した．また，類似度計算回数を図 6 に示した．高精度化法は $|\psi| < 2000$ 程度までは最速だが，それ以上になると毎時刻更新法と順位が入れ替わっている，これは $|\psi|$ が増加することによって，クエリとデータ間の共通要素候補数が減少し，高精度化のオーバーヘッドが大きくなるためである．$|\psi|$ が増加することによる共通要素候補数の減少は 共通要素候補数 = 積集合サイズ であることから，表 1 より確認することができる．以上のことから毎時刻更新は類似しているデータと類似していないデータが混在するデータベースから類似しているデータを検索する場合に効果を発揮する手法であり，高精度化法は似通ったデータを多く持つデータベースに対する類似検索に効果を発揮する手法であると言える．一方で提案手法の実行時間は，$|\psi|$ の小さな場合には，高精度化法の実行時間に近い値であり，$|\psi|$ の大きな場合には，毎時刻更新法に近い値であった．これは，上限値の高精度を適応的に行うことで上限値の高精度化のオーバーヘッドの増加を抑えたうえで，毎時刻更新法が低速化する平均類似度が高い場合でも検索を高速化できていることを表している．つまり，データベースから一部の類似するデータを検索する検索問題において，$|\psi|$ の値によ

らず, 効果的な手法が実現できたことが分かった.

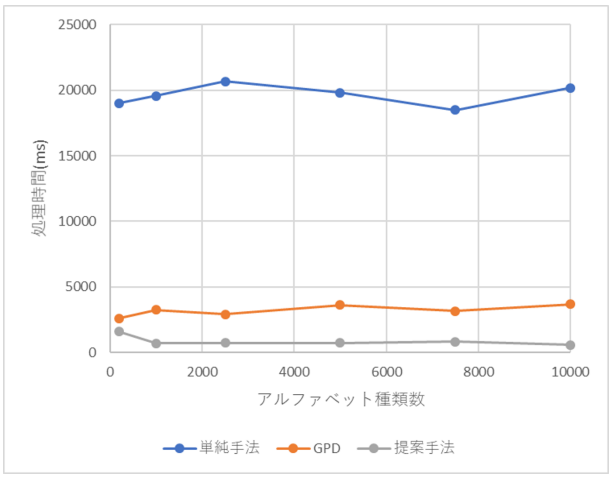

図 4　$k = 10$ での単純手法, GPD との比較

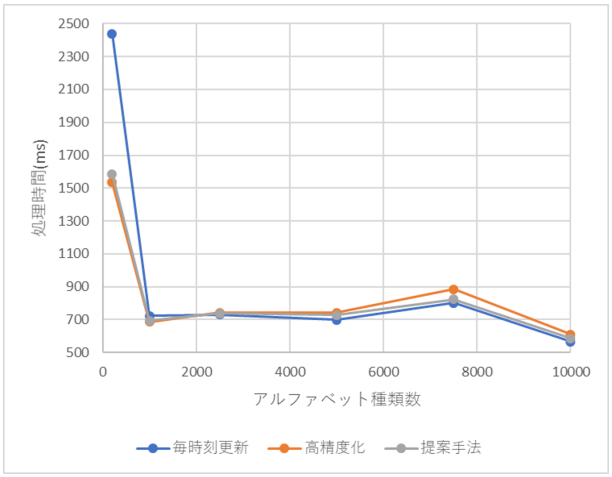

図 5　$k = 10$ での適応的な上限値の高精度化

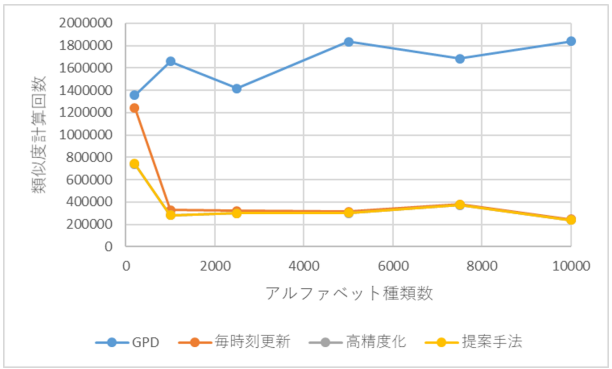

図 6　$k = 10$ での類似度計算回数

表 1　積集合サイズ

アルファベット種類数 $\|\psi\|$	200	1000	250	5000	7500	10000
第 10 位	63	51	54	41	35	52
21 位以下の平均	54	14	8	3	2	2

5.2　実データによる実験

以下の 2 種類のデータセットを対象として検索実験を行った.

- Market Basket dataset [8]
- Click Stream data set [9]

いずれのデータセットも先行研究 [5] で用いられたものである. 実データの検索実験におけるストリームデータの作り方は, IBM Quest data generator による実験と同様である. 二つの実データに対してパラメータ $k = 10, 25, 50, 75, 100$ に変化させた時の実行時間を測定した.

5.2.1　Market Basket dataset

Market Basket dataset は, 複数の買い物客の購入品リストを集めたものである. 以下にデータセットのパラメータを示す.

- 商品 (アルファベット) の種類 $|\psi| = 16470$
- データ (データパターン) 数 $p = 88162$

また, データベース内の集合の要素数の平均は 10.3 であったため, スライディングウインドウのサイズ $w = 10$ とした.

このデータセットでは, アルファベット種類数 $|\psi|$ とデータパターン数 p が大きいため, データベース内の集合との共通要素候補数が少ない. そのため, 高精度化法の効力が低下し, 毎時更新法が最も高速になると予想できる. よって提案手法は高精度化法よりも毎時更新法に近い値になると期待した.

実験結果を図 7 に示した. 提案手法は単純手法, GPD に比べ高速であった. 一方で, 提案手法は共通要素がほとんどないため毎時刻更新法に近い値になるだろうという期待に反して, 高精度化法とほとんど同じ値になってしまった.

5.2.2　Click Stream data set

Click Stream data set は, MSNBC のウェブサイトでユーザーがクリックした URL を順に記録したものであり, 各 URL は “news” や “tech” のようにカテゴリに量子化されている. このデータセットのパラメータは以下のとおりである.

- カテゴリ (アルファベット) の種類 $|\psi| = 17$
- データ (データパターン) 数 $p = 31790$

データベース内の集合の要素数の平均は 13.33 であったため, スライディングウインドウのサイズ $w = 13$ とした. このデータセットでは, データパターン数 p は大きいものの, アルファ

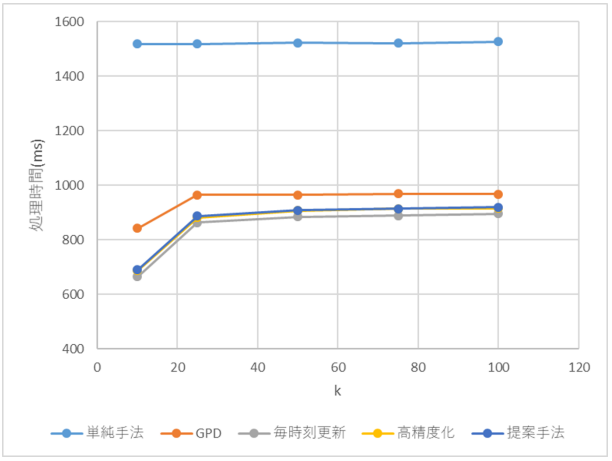

図 7　Market Basket dataset での処理時間

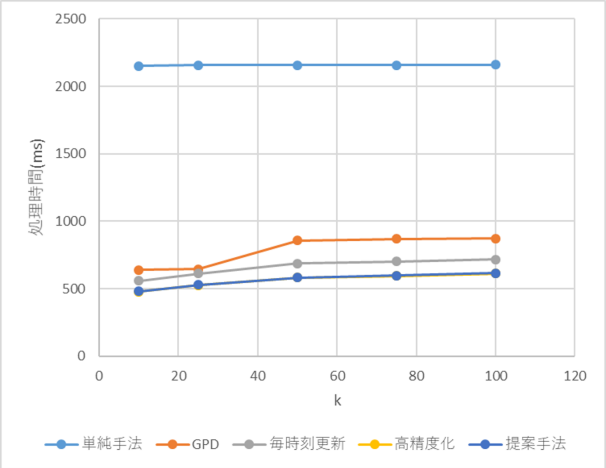

図 8　Click Stream data set での処理時間

ベット種類数 ψ が少ないため, クエリとデータベース内の集合が共通要素候補を多く持つデータセットになっている. そのため, 共通要素候補判定の効果が大きく, 高精度化法が最も高速になると予想できる. よって提案手法は毎時更新法よりも高精度化法に近い値になると期待した.

実験結果を図 8 に示した. こちらも, 提案手法が単純手法, GPD に比べ高速であった. また, Market Basket dataset とは異なり提案手法は期待通り高精度化法に近い実行時間であった.

5. 2. 3　実データによる実験まとめ

どちらのデータセットでも提案手法と高精度化法の処理時間が非常に近い, この原因として共通要素候補判定にかかるコストを低めに見積もりすぎている事が考えられる. 実データによる実験では人工データによる実験と比べて, ウィンドウサイズ w が小さい. このため, 共通要素候補判定以外の処理コストが大きく影響した可能性がある. しかし, そのコストの原因の詳細を特定できなかった.

6　ま　と　め

本研究では, 従来のクエリがデータストリームであり, データベースは複数の静的な集合で構成される CEQ 問題とは逆に, クエリが静的な集合で, データベースに複数のデータストリームが登録されている状況で, クエリと類似した集合を検索する CED 問題を取り扱った.

我々は, この CED 問題に対し, CEQ 問題に対する枝刈アルゴリズムを拡張したうえで, 更に枝刈り効率を上げるため, 類似度の上限値をより厳密に求める次の 2 手法を考案した.

（1） 最後に類似度を厳密計算した時刻からの経過時間に応じて, 上限値を段階的に増やす手法.

（2） スライディングウィンドウから離脱する要素は既知であることに着目し, クエリ集合 Q とデータベースの集合 S の間の共通要素数をより厳密に推定する手法

これらを用いて, ボトルネックである類似度計算回数を減少させて, 計算時間の削減を図った. また、この 2 手法を適応的に用いることによって多くの場合においての, 高速な手法の確立を目指した.

検索実験により提案手法の評価を行った. その結果, 提案手法が, 従来手法に比べ高速であることが確認できた.

一方で 各手法のコストの見積もりが不十分だと考えられる結果も確認できた. よって, それを断定する手法の確立が今後の課題である.

謝　　辞

本研究は科研費基盤研究 (C)18K11311 の助成を受けたものである.

文　　献

[1] Leong Hou U, Junjie Zhang, Kyriakos Mouratidis, and Ye Li, “Continuous Top-k Monitoring on Document Streams”, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, vol.29, issue.5, May 1 2017

[2] Kyriakos Mouratidis, Spiridon Bakiras, Dimitris Papadias, “Continuous monitoring of top-k queries over sliding windows”, SIGMOD ’06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, June 2006, Pages 635–646

[3] Di Yang, Avani Shastri, Elke A. Rundensteiner, Matthew O. Ward, “An Optimal Strategy for Monitoring Top-k Queries in Streaming Windows”, Conference: EDBT 2011, 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21-24, 2011

[4] Christodoulos Efstathiades, Alexandros Belesiotis, Dimitrios Skoutas, Dieter Pfoser. “Similarity Search on Spatio-Textual Point Sets“ EDBT, pages 329-340, 2016.

[5] X. Xu, C. Gao, J. Pei, K. Wang, and A. Al-Barakati, “Continuous similarity search for evolving queries,” Knowledge and Information Systems, vol.48(3), pp.649-678, September 2016.

[6] T. Yamazaki, H. Koga and T.Toda, “Fast Exact Algorithm to Solve Continuous Similarity Search for Evolving Queries”, in Proc. Asia Information Retrieval Symposium (AIRS2017), springer LNCS 10648, pp.84-96, 2017.

[7] https://sourceforge.net/projects/ibmquestdatagen/

[8] http://fimi.ua.ac.be/data/

[9] http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php