D-011

集合間類似度を用いたストリームデータの top-k 類似検索における枝刈アルゴリズムの改善

野口 大樹 †　　古賀 久志 †　　戸田 貴久 †

1　はじめに

近年，テクノロジーの進化によって SNS やネットニュースであったり，センサーなどの機器から流れてくるデータなど，無限に増え続けるストリームデータの解析の重要性が増してきている．本論文では，その中でもストリームデータに対する類似検索に着目する. ストリームデータに対する類似検索は，インターネットサービス等でのユーザの嗜好に適した情報推薦や機器の異常検知などの応用を持つ．例えば, ウェブサイトで広告を表示したい場合，ユーザの検索履歴や広告のクリック率等はストリームデータであるが, これはユーザの好みを示す．そして，ストリームデータをクエリとして広告を対象とした類似検索を行うことでユーザの好みに合った広告を表示することができる.

ストリームデータに対する類似検索には様々な問題設定が存在するが，その中で本研究では, Xu ら [1] によって提唱された「Continuous Similarity Search for Evolving Queries」(CSEQ) 問題 [1] を取り上げる．この問題は，要素がアルファベットであるデータストリームのスライディングウインドウ内の要素集合をクエリとして，集合のデータベースからクエリと最も類似している上位 k 個のデータ (集合) を探索することを目的とする．この問題では，毎時刻，データストリームには新しい要素が追加され，それに伴ってクエリが変化する．その結果，上位 k 個のデータを毎時刻更新する必要がある．この問題に対する自明な解法は，クエリが更新される度にクエリとデータベース内の全データとの間で類似度計算を行うことである. しかし，類似度計算には時間がかかるため，類似度計算の計算回数が多くなることは好ましくない. この問題に対しては，2 タイプの高速化手法が提案されている．1 つ目は過去の類似度の値から現在時刻での類似度の上限値を求め，上位 k 個になり得るデータのみ類似度計算を行う枝刈法 [1] である．2 つ目は類似度が変化するデータを転置インデックスを用いて高速発見し，それらに対してのみ類似度を求める山崎らの手法 [2] である．山崎らによる手法は枝刈法よりはるかに高速であることが報告されている.

CSEQ 問題ではクエリが動的に変化し，データベース内の集合は不変であるが，我々のプロジェクトではクエリが不変でデータベース内の集合が動的に変化する「Continuous Similarity Search for Evolving Database」(CSED) 問題を取り扱うことを計画している．CSED 問題ではデータベース内の集合が毎時刻変化するため，転置インデックスは更新オーバーヘッドが大きく不向きである．そこで，転置インデックスを使わない枝刈法を用いて CSED に対する高速アルゴリズムを実現することを検討している．本論文では，枝刈法を高速化する要素技術を確立するため，既存の CSEQ 問題に対する枝刈り法を高速化することを研究目的とする．とくに，類似度の上限値をより厳密に求めて類似度計算回数を削減し，さらに山崎らの手法で使用された高速な類似度更新法を組み込むことで，転値インデックスを用いることなく枝刈り法を高速化することに成功した．本論文で提案する手法は CSED 問題においても有効であると考えられる.

本論文では，第 2 章で CSEQ 問題を定義し，第 3 章で従来手法，第 4 章で改良手法について述べる．第 5 章で改良手法を実験的に評価し，第 6 章で結論を述べる.

2　問題定義

Continuous Similarity Search for Evolving Queries(CSEQ) 問題 [1] の定義を説明する．$\psi = \{x_1, x_2, \cdots, x_{|\psi|}\}$ をアルファベットとする．データストリーム 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の集合 $\{S_1, S_2, \cdots, S_n\}$ で構成される．D 内の集合は時間によって変化せず静的である.

CSEQ 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 から検索する問題である．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

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

を用いる.

3　従来手法

本節では，CSEQ 問題に対する既存アルゴリズムを 3 つ紹介する.

† 電気通信大学大学大学院情報理工学研究科
〒 182−8585 東京都調布市調布ヶ丘 151

3.1 単純手法

CSEQ 問題に対する最も単純な解法は，時刻 t にクエリ Q_t とデータベース D 内の全ての集合 S_i $(1 \leq i \leq n)$ との類似度を計算し top-k を求める手法である．この単純解法では，各時刻で類似度の計算を n 回行うことがボトルネックとなる．2 集合間の Jaccard 係数 1 回の時間計算量は 2 つの集合の要素数の合計に比例するため，1 回の類似度計算にかかる時間は $O(|Q_t| + |S_i|) = O(W + |S_i|)$ となる．よって，各時刻での時間計算量は少なくとも $\Omega(nW)$ になる．

3.2 枝刈による厳密解法

この節では，Xu らによる従来手法 GP(general pruning-based method)[1] について述べる．GP は，D の一部の集合 S_i 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 t の top-k 類似度の下限値を下回る場合に，類似度計算を省略することで実行速度を高速化する．

3.2.1 類似度の上限値

時刻 t に計算されたクエリ Q_t と集合 S_i の類似度が $sim(S_i, Q_t)$ であったとき, 時刻 u 経過後の類似度の上限値を $sim(S_i, Q_t)^{+u}$ とする。時刻が u 経過すると, クエリ Q_t の要素が u 個入れ替わるから, 時刻 $t+u$ の類似度の上限値は

$$\mathrm{sim}(S_i, Q_t)^{+u} = \frac{|S_i \cap Q_t| + u}{|S_i \cup Q_t| - u} \tag{1}$$

$$= \frac{|S_i \cap Q_t| + u}{|S_i| + |Q_t| - |S_i \cap Q_t| - u} \tag{2}$$

$$= \frac{(|S_i| + |Q_t| + u) \cdot \mathrm{sim}(S_i, Q_t) + u}{|S_i| + |Q_t| - u \cdot \mathrm{sim}(S_i, Q_t) - u} \tag{3}$$

となる．式 (3) は上限値が $\mathrm{sim}(S_i, Q_t)$ から高速で求められることを示している．

ここで GP は S_i に対応して，上限値 $\mathrm{sim}(S_i, Q_t)^{+n}$ 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 min_step_i を求める.min_step_i は $\mathrm{score}(topk^t) < \mathrm{sim}(S_i, Q_t)^{+min\_step_i}$ を満たす最小の整数である．GP では時刻 t から $t + min\_step$ における S_i の上限値を $\mathrm{sim}(S_i, Q)^{+min\_step}$ と設定する．

3.2.2 top-k の算出方法

時刻 T において，データベース D を集合群 $R = \{S_i | t_i + min\_step_i = T\}$ とそれ以外 $D \backslash R$ に分割する.t_i は S_i に対して最後に類似度を計算した時刻である．R に含まれる集合は類似度が時刻 t_i での $\mathrm{score}(topk_i^t)$ を超えて，十分大きくなっている可能性があるので類似度を再計算し，R の中で top-k を求める．この時, k 番目に類似した集合の類似度は top-k に対する類似度の下限値 lb となる．$S_i \in D \backslash R$ のうち, 上限値が lb を下回る集合に対しては類似度の計算を省略する．lb の更新は逐次行う．

3.3 転置インデックスを使用する方法

山崎らは転置インデックスを用いて時刻が 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に対して類似度計算を省略する手法を提案した．この方法では時刻が進んだときにスライディングウインドウに入る要素 IN とスライディングウインドウからでる要素 OUT をどちらも含まない集合の類似度が不変であるという性質を利用し，それらの集合に対して類似度計算を省略する．逆に処理対象となる IN または OUT を含む集合を転置インデックスで高速に見つける．また類似度が変化する集合に対しては類似度 $\mathrm{sim}(S, Q_{t+1})$ を，時刻 t の類似度 $\mathrm{sim}(S, Q_t)$ を $O(1)$ で更新して高速に求める．

4 枝刈手法の改善

本章では, 類似度計算回数を減らすために, 類似度の上限値をより正確に求める手法を 2 つ，類似度計算を高速化する手法を 1 つ提案する．

4.1 段階的な上限値の更新

GP では, 時刻 t において集合 S_i の min_step が min_step_i となった場合, 類似度の再計算が行われない限り, 時刻 t から $t + min\_step_i$ における S_i の上限値は常に $\mathrm{sim}(S_i, Q)^{+min\_step_i}$ と設定される．このやり方は上限値を時刻 t から $t + min\_step_i$ までの間に更新しないため, 記憶しておいた変数を参照するだけで, 上限値を取り出すことができるという利点がある. しかし, 時刻 $t + \alpha$ $(1 \leq \alpha \leq min\_step_i)$ の上限値 $\mathrm{sim}(S_i, Q)^{+\alpha}$ は $\mathrm{sim}(S, Q)^{+min\_step}$ よりも低い. つまり, 必要以上に高い値になっているため, 類似度計算回数の増加につながる危険性がある．

$\mathrm{sim}(S, Q_t)^{+u}$ は $|S \cap Q_t|$ が分かれば式 (2) より加算が 2 回, 減算が 2 回, 除算が 1 回で $O(1)$ で計算できる．よって, 上限値の精度を高めるため上限値の毎時刻更新を行う．

4.2 離脱したデータに注目した上限値の高精度化

GP 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, 将来離脱するデータ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り，提案手法ではその離脱するデータの情報を利用して上限値を厳密に求める．

4.2.1 上限値の高精度化

クエリから離脱したときに積集合の大きさを下げるものを共通要素と定義する．時刻 t のクエリ Q_t の要素 $q_1 \sim q_u$ の中に共通要素が n 個あった場合, 時刻 $t+u$ の上限値の式は

$$\frac{|S \cap Q_t| + (u - n)}{|S \cup Q_t| - (u - n)} \tag{4}$$

とすることができる．

また, min_step は上限値が現在の top-k の値を超える最小時刻であるが, 図 1 に示すように上限値が n ステップ上昇しなくなるため $min\_step = (min\_step + n)$ とす

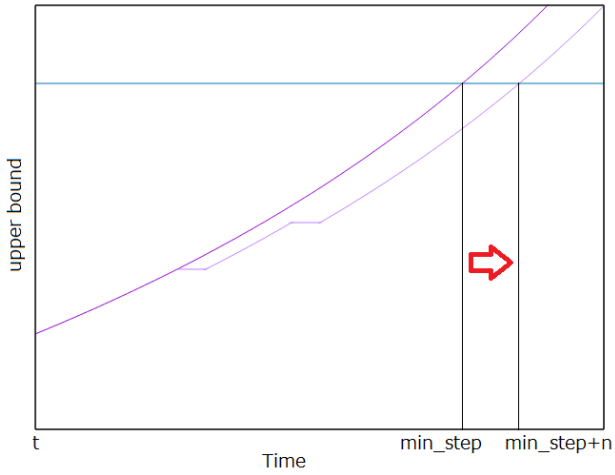

図 1　上限値の高精度化による変化

ることができる．つまり，jaccard 計算を再計算するタイミングを遅らせることになるので類似度計算回数を削減できる．

4.2.2　共通要素の判定

クエリ Q_t の各要素が共通要素であるかどうか判定するために, Q_t をアルファベットのヒストグラムで表現する. 時刻が変化した時の Q_t の更新はヒストグラムで表現されている場合，$O(1)$ と高速に実行できる．

クエリに同じアルファベットが複数あった場合, 離脱したときに積集合の大きさを下げるのは, 新しくスライディングウインドウに追加された要素である．よって共通要素の判定は, クエリの持つアルファベット各種類について, ヒストグラムを比較して $min\{$ クエリの保有数 , データの保有数 $\}$ だけ新しいものから共通要素とすればよい．

4.3　カウンタを用いた jaccard 係数計算の高速化

山崎らの手法 [2] では，類似度の値が変化する可能性がある集合に対して，前時刻の共通要素数 $|S \cap Q|$ の値を再利用して高速に類似度更新をしている．一方，Valari らはグラフのノード間の類似度検索において，類似度計算が行われる可能性が高いノードペアに対して共通隣接ノード数をカウンタとして記録し類似度計算を高速化している [3]．また、4.1 節でも述べたように $|S \cap Q|$ の値が分かれば上限値も高速で求めることができる．

$|S \cap Q|$ の値は jaccard 係数を計算する時に求まっている。そこで本研究では、近い将来に類似度計算が行われる可能性が高い集合に対して $|S \cap Q|$ の値を記憶して計算の高速化を行う．近い将来に類似度計算が行われる可能性が高い集合は，min_step の値が小さな集合である．min_step の値が小さな集合に対して $|S \cap Q|$ の値をカウンタとして記録し $min\_step = 0$ となるまで離脱したデータと挿入されたデータに着目してカウンタを更新する。

5　実験結果

従来手法 GP, 提案手法のうち段階的な上限値の更新を行う枝刈法 GPUE, GPUE に加えて離脱したデータに注目した上限値の高精度化を行う枝刈法 GPD，GPD に対して $min\_step <= 5$ の集合の類似度計算にカウンタを用いた GPCL，GPD に対して全ての集合の類似度計算にカウンタを用いた GPCA を実装し性能を比較する．各時刻 t において, クエリ Q_t 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 $w = 100$, アルファベット種類数 $|\psi| = 200, 1000$ という条件の下でデータベースの集合数 n を変化させて行った. また, 求める top-k リストのサイズ $k = 100$ とした．

1. $|\psi| = 200$ で n を変化させた場合の実行時間
   実行結果を図 2 に示した．GP，GPUE，GPD の内 GPD が最も高速であった．アルファベット種類数が小さいことで，平均集合間類似度が大きくなり，共通要素がクエリから離脱する可能性が高くなる．よって離脱したデータに注目した上限値の高精度化の効果が大きくなったためであると考えられる．また，カウンタを用いることでさらに高速化できることが分かったが，近い将来，類似度が計算される集合にのみ適応した GPCL は効果が薄く，カウンタを使う範囲を限定せず全ての類似度計算をカウンタで行う GPCA が効果が高いことが分かった．
2. $|\psi| = 1000$ で n を変化させた場合の実行時間
   実行結果を図 3 に示した．GP，GPUE，GPD の内 GPUE が最も高速であった．アルファベット種類数が大きいことで，平均集合間類似度が小さくなり，共通要素がクエリから離脱する可能性が引くなる．よって共通要素を発見するオーバーヘッドのみが増大したためであると考えられる．カウンタの効果は $|\psi| = 200$ の時と同様であった．

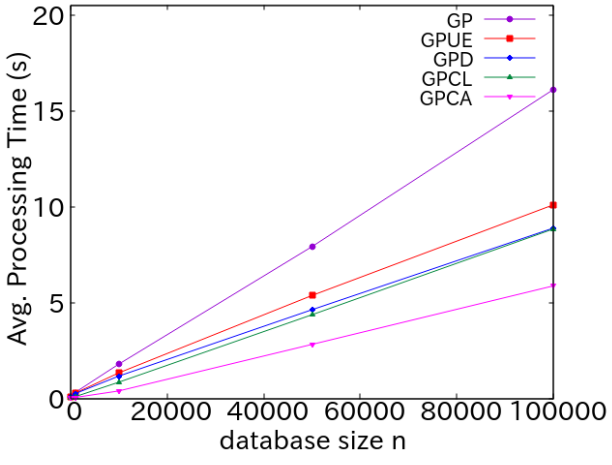

図 2　$|\psi| = 200$ で n を変化させた場合の実行時間

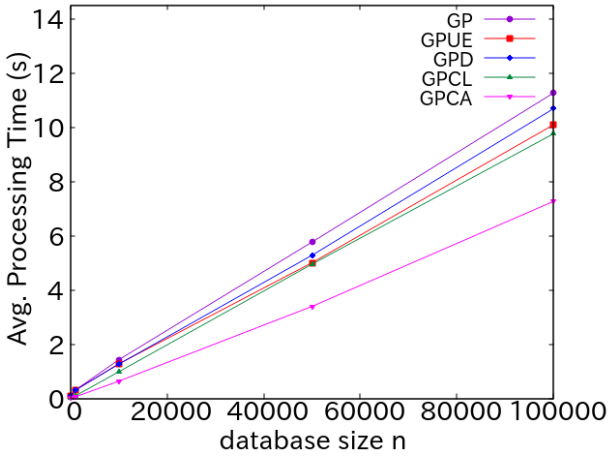

図 3　$|\psi| = 1000$ で n を変化させた場合の実行時間

6　まとめ

本研究では CSEQ 問題における枝刈アルゴリズムを取り扱った．この問題に対しては，転置インデックスベースの高速な手法が既に存在するが，CSED 問題に対しては不向きである．そこで CSED 問題にも適用可能な枝刈りベースのアルゴリズムの改良手法を研究対象とした．

本研究では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毎時刻更新法．
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離脱データ法

．これらを用いて上限値をより高精度に算出し，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る．また、共通要素のカウンタを用いることで，類似度計算の高速化を図った．

検索実験により提案手法の評価を行った．その結果，毎時刻更新法が多くの場合において，先行研究で提案された枝刈り手法 GP を上回ることを示せた．また，離脱データ法は，Q とデータベース内の集合間の平均類似度が大きい場合に毎時刻更新法と組み合わせることにより，毎時刻法単体よりも性能改善ができることが分かった．カウンタは近い将来に類似度計算を行うものに用いることで高速化すると予想したが，全ての類似度計算に適用するとより高速になることが判明した．

また，今回の研究を通して，適切な枝刈り手法は与えられたデータベースの性質によって変わることがわかった．データベースの性質を学習して，枝刈り手法を適応的に変える機構の実現することも大切である．

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．

参考文献

[1] 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.

[2] 山崎智博・古賀久志・戸田貴久 ”集合間類似度を用いたストリームデータの top-k 類似検索に対する高速な厳密解アルゴリズム”, 信学技報 COMP2017-1, pp.1-9, 2017.

[3] Elena Valari，Apostolos N. Papadopoulos ”Continuous Similarity Computation over Streaming Graphs”，ECML PKDD 2013: Machine Learning and Knowledge Discovery in Databases pp 638-653.