複数ストリーム環境における Top-k 共起パターンマイニング

天方　大地† 　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: †{amagata.daichi,hara}@ist.osaka-u.ac.jp

あらまし　ビッグデータおよび IoT 時代における様々なアプリケーションでは，オブジェクトはストリーム形式で発生している．ストリームから重要なパターンをリアルタイムにマイニングする技術は，そのようなアプリケーションにおいて必要不可欠である．本稿では，複数のストリームに現れるオブジェクト集合をモニタリングする問題に取り組む．具体的には，あるオブジェクト集合は，その集合が現れるストリームの数によってランキングされ，その上位 k 個のものをモニタリングする．この問題を正確かつ効率的に解くため，グラフと転置ファイルを組み合わせたデータ構造である CP-Graph，および CP-Graph を利用したアルゴリズムを提案する．CP-Graph は，与えられたパターンがどれだけのストリームに現れるかを効率的に計算し，正解集合に含まれないパターンを枝刈りしながら解を更新できる．実データを用いた実験により，提案アルゴリズムの有効性およびスケーラビリティを確認する．

キーワード　ストリーム，共起パターン，マイニング

1. はじめに

パターンマイニングは，多くのドメインにおける基本的な問題であり，市場分析 [10]，クリックストリーム [8]，ネットワークモニタリング [5] などの幅広いアプリケーションに用いられている．ビッグデータおよび IoT 時代における様々なアプリケーションでは，オブジェクトはストリーム形式で発生しており，ストリーム環境におけるパターンマイニング問題に関する研究は盛んに行われている．本稿では，複数のストリームが存在する環境を想定し，Top-k 共起パターンマイニング問題に取り組む．

研究動機．複数のストリーム（$\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_{|\mathcal{S}|}$）が継続的にトランザクション（オブジェクトの集合）を生成し，それらはスライディングウィンドウによって管理されているとする．ここで，P をオブジェクトの集合（パターン）とし，P が現れたストリームの数を $\mu(P)$ とする．$\mu(P) \geq 2$ である（かつ $\mu(P) = \mu(P')$ となる $P' \supset P$ が存在しない）場合，P を（排他的）共起パターン (注1) と呼ぶ．μ が最大である k 個の（排他的）共起パターンをモニタリングする問題も幅広いアプリケーションを持つ．

例 1．ソーシャルネットワークサービス（SNS）の各ユーザは，一つのストリームと見なせる．例えばある Twitter ユーザのツイートは，継続的に発生が起きるストリームである．多くのユーザのツイートに同じキーワードが現れる場合，そのキーワードを共起パターンと見なせる．つまり，本問題を解くことにより，新出した話題のリアルタイムな発見が可能となる．■

Web ページの遷移パターンや e コマースでの購買パターンのマイニングも例 1 と同様に本問題のアプリケーションとして挙げられる．

ここで，ユーザが閾値 ρ を与えた時，ρ 以上のストリームで現れる共起パターンをモニタリングする問題も考えられる [12]．この問題も上で挙げたアプリケーションに適応できる．しかし，ユーザにとって ρ を指定することは難しく，適切な ρ を事前に知ることは非現実的である [8]．例えば，ρ が小さいと解のサイズが大きすぎる可能性があり，ρ が大きいと解のサイズが 0 となる可能性がある．一方，k を指定することにより，ユーザは希望の数，かつ最も共起しているパターンをモニタリングできる．それにも関わらず，本問題はこれまでに研究されていない．

課題．複数ストリームにおける排他的共起パターンマイニング問題は，以下の事実によりチャレンジングである．

（1）リアルタイム性．例 1 に表されるように，本問題のアプリケーションはリアルタイム性が要求される．本問題に対する単純な解法は，トランザクションが発生・消滅した時，全てのパターンの μ を計算し，ソートにより上位 k 個の排他的共起パターンを抽出することである．しかし，パターンの数は指数的に増える [7] ため，この解法は現実的でない．

（2）動的な μ と閾値．あるパターン P の μ（$\mu(P)$）は，トランザクションの発生および消滅により変化する．そのため，現時点で共起パターンでない，または排他的でないパターンも将来的に排他的共起パターンになる可能性がある．同時に，現時点で解に含まれる排他的共起パターンも，μ が小さくなることにより解に含まれなくなる可能性がある．

（3）効率的な枝刈りのためのデータ構造の必要性．上の 2 つの課題から，不必要なパターンの枝刈り，および与えられたパターンの μ を効率的に計算するためのデータ構造が必要である．既存のデータ構造は，静的データ [13] や単一のストリーム [7] のみ考慮しているため，適用可能でない．さらに，閾値以上の共起パターンをモニタリングする問題に対するデータ構造 [12] も，本問題に対しては効率的でない．本問題では閾値（上位 k 番目の μ）が事前に知り得ないため，既存アルゴリズムの単純な適用方法は，$\rho = 2$ とすることである．これは，全てのパター

（注1）：(closed) co-occurrence pattern

ンを列挙する単純な解法と同義であり，非現実的である．

提案アルゴリズムの概要．リアルタイム性を満たすために最初に解決すべき課題は，与えられたパターンの μ を高速に計算することである．これを実現するため，新たなデータ構造である CP-Graph を提案する．CP-Graph は，トランザクション中のオブジェクト集合をグラフで管理し，効率的にパターンを表現する．例えば，あるパターン $\{o_i, o_j\}$ が存在する場合，o_i および o_j は頂点（v_i および v_j）と見なされ，その間には辺が存在する．さらに，各頂点 v_j は転置ファイルを保持する．この転置ファイルは，o_j を含むパターンの μ の上界値，および正確な値を計算するために用いられる．CP-Graph の利点として，$\mu(\{o_i, ..., o_j\})$ は，v_j の転置ファイルをチェックするだけで得られる点が挙げられる．

次に解決すべき問題は，新たに解に含まれるパターンを効率的に検索することである．本問題における閾値は動的に変化するため，スライディングウィンドウ上で管理しているトランザクションが更新された場合，閾値を推定し，無駄な検索を削減する必要がある．この閾値の推定，および解の更新を CP-Graph を用いて効率的に実行する．実データを用いた実験により，提案アルゴリズムの有効性およびスケーラビリティを確認する．

本稿の構成．以下では，2. 章で本稿の問題の定義および性質を紹介し，3. 章で関連研究について述べる．4. 章で本稿の提案アルゴリズムについて説明し，5. 章で性能評価のために行った実験の結果を紹介する．最後に 6. 章で本稿をまとめる．

2. 予 備 知 識

本稿では，複数のストリーム（$\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_{|\mathcal{S}|}$）が存在する環境を想定する．各ストリーム $\mathcal{S}_i$ は，継続的にトランザクションを生成する．あるトランザクション t_j は，ストリームの識別子とオブジェクト集合のタプルである．例えば，$\mathcal{S}_i$ が t_j を生成した場合，$t_j = \langle i, O \rangle$ である．ここで，O 内のオブジェクトは辞書順でソートされていると想定する [7]．つまり，$o_i \in O$ および $o_j \in O$ が与えられた時，$o_i \prec o_j$ である．本問題を定義するために，本稿で用いるシンボルおよび用語を以下で定義する．

定義 1（$\mu(P)$）．$P = \{o_p, ..., o_q\}$ をオブジェクト集合（パターン）とすると，$\mu(P) = |\{\mathcal{S}_i \in \mathcal{S}\}|$ である．ここで，$\mathcal{S}$ はストリームの集合であり，$\mathcal{S}_i$ は P を含むトランザクションを生成したストリームである．

定義 2（共起パターン）．$\mu(P) \geqq 2$ であれば，P は共起パターンである．

本問題では，*closed*（排他的）の概念を導入する [3], [11]．P および P の上位集合 P' が与えられたとする．$\mu(P) = \mu(P')$ であれば，P と P' を共に出力することは無駄である．なぜなら，P が解に含まれていない，かつ P' が解に含まれていれば，$\mu(P) = \mu(P')$ と分かるからである．

定義 3（排他的共起パターン）．P を共起パターンとする．$\mu(P) = \mu(P')$ かつ $P \subset P'$ となる P' が存在しなければ，P は排他的共起パターンである．

表 1　トランザクション集合

トランザクション	ストリーム	オブジェクト集合	生成時刻
t_1	$\mathcal{S}_1$	$\{o_a, o_b, o_x, o_y\}$	c'
t_2	$\mathcal{S}_2$	$\{o_d, o_x, o_y\}$	c'
t_3	$\mathcal{S}_3$	$\{o_a, o_c, o_d, o_x, o_y\}$	c'
t_4	$\mathcal{S}_4$	$\{o_a, o_b, o_d, o_y\}$	c'

本稿では，ストリームデータの管理モデルとして，スライディングウィンドウ [2] を用いる．ここで，w をウィンドウサイズとすると，w は管理する最新のデータ数（例えば 1,000,000 個のデータをウィンドウに含める），または時間の長さ（例えば現在時刻から 2 時間以内に発生したデータをウィンドウに含める）として定義される．さらに，Δ をスライドサイズとする．w がデータ数を表す場合，Δ は定数であり，w が時間の長さを表す場合，$\Delta = 1$ [time-unit] である．本稿の提案アルゴリズムはどちらのモデルにも適用可能であるため，以降の説明では w は時間の長さとし，c_{now} を現在時刻とする．

問題定義．k，w，および $\mathcal{S}$ が与えられたとする．本問題は，μ が最大となる k 個の排他的共起パターンをモニタリングすることである．

例 2. スライディングウィンドウ上で管理されているデータを表 1 に表す．o_y は 4 つのストリームで現れているため，$\mu(\{o_y\}) = 4$ である．同様に，$\mu(\{o_b\}) = 2$ であるが，$\mu(\{o_b, o_y\}) = 2$ であるため，$\{o_b\}$ は排他的共起パターンでない．$k = 3$ とすると，c_{now} における正解集合は，$\{o_y\}$，$\{o_a, o_y\}$，および $\{o_d, o_y\}$ である．（最後の 2 つのパターンについては，$\{o_x, o_y\}$ と交換可能である．）■

トランザクションは継続的に生成されるため，スライドごとの解の更新は正確かつ高速であることが求められる．これを実現するアルゴリズムを設計するため，本問題の性質を紹介する．

性質 1. スライディングウィンドウ上に現れる全てのパターンを列挙することは，#P 完全である [9].

しかし，本問題は他のパターンマイニング問題 [1], [6] と同様に，アプリオリ（apriori）性質を持つ．つまり，2 つのパターン P および P' について，$P \subset P'$ であれば $\mu(P) \geqq \mu(P')$ である．これにより性質 1 を緩和できると同時に，性質 2 が得られる．

性質 2. P を排他的共起パターンとする．P が解に含まれなければ，P の上位集合も解に含まれることはない．

性質 2 により探索空間を削減できるが，本問題は以下の課題を持つ．

性質 3. P を共起パターンとし，$t_a = \langle i, O \rangle$（$P \subseteqq O$）について考える．(i) 新しいトランザクション $t_b = \langle i, O' \rangle$（$P \subseteqq O'$）が生成された場合においても，$\mu(P)$ が増加するとは限らない．(ii) t_a がウィンドウから除外されても，$\mu(P)$ が減少するとは限らない．

証明．ページ数の制限により，全ての証明を割愛する．　□

性質 4. P_1，P_2，...，および P_k を c_{now} における Top-k 排他的共起パターンとする．P_i（$i \in [1, k]$）が消滅しない場合や

$\mu(P_i)$ が減少しない場合においても，上位 k 番目の μ（τ_{now}）が $c_{now}+1$ において減少する場合がある．

3. 関 連 研 究

静的データやストリームデータに対するパターンマイニング問題は多くの関心を集めており，これまでに多数のアルゴリズムが提案されている．本稿では複数ストリーム環境におけるパターンマイニングに関する既存研究 [4], [12] に焦点を当てる．

文献 [4] では，H-Stream と呼ばれるアルゴリズムが提案されている．しかし，このアルゴリズムは正確な解をモニタリングすることを保証していない．文献 [12] では，Seg-tree と呼ばれるデータ構造，および CooMine と呼ばれるアルゴリズムを提案している．（詳細は文献 [12] を参照．）図 1 は，表 1 におけるトランザクション集合から構成される Seg-tree を簡潔に示している．トランザクションが与えられた時，最もマッチ数が多いプレフィックスノードに対して差分のノードを追加し，Seg-tree の各テイル（tail）ノードが，対応するトランザクションを保持する．ユーザが閾値 ρ を与えた時，CooMine は ρ 以上のストリームに現れるパターンを，Seg-tree を用いて高速に検索する．CooMine は Top-k モニタリングを考慮しておらず，1. 章で述べた通り，$\rho=2$ とすることが単純な適用方法である．また，CooMine は μ の更新を考慮していないため，本稿の問題を効率的に解決しない．これを 5. 章で明らかにする．

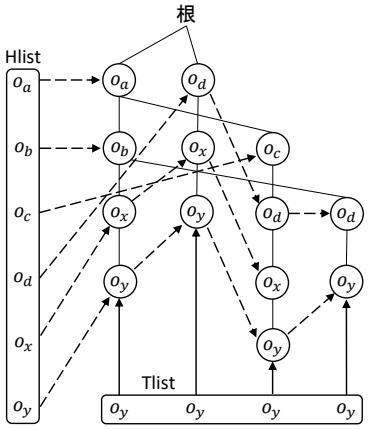


図 1　表 1 におけるトランザクション集合から構成される Seg-tree

4. 提案アルゴリズム

提案アルゴリズムは，不必要なパターンの枝刈りやインクリメンタルな解の更新を実現するために，ウィンドウ上のトランザクションをインメモリ型インデックスで管理する方針をとる．提案するインデックスは，CP-Graph（Co-occurrence Pattern Graph）と呼ばれ，CP-Graph を利用したアルゴリズムにより解の更新を行う．ここで，$\mathcal{A}$ を正解集合とし，$\mathcal{A}$ の各要素は，排他的共起パターンとその μ のタプルである．$\mathcal{A}$ の更新アルゴリズムは，ウィンドウの各スライドにおいて，以下のステップを順に行う．

ステップ 1（インデックスの更新）．新たなトランザクションを CP-Graph に反映させ，ウィンドウから除外されたトランザクションを CP-Graph から取り除く．（4.2 節）

ステップ 2（$\mathcal{A}$ の更新）．$c_{now}-1$ における解と CP-Graph から，c_{now} における k 番目に大きい μ（閾値）を推定し，その閾値を用いて $\mathcal{A}$ を更新する．

4.1 CP-Graph

解を高速に更新するためには，必要な排他的共起パターンの列挙とその μ の計算を効率的に行う必要がある．CP-Graph はこの要件を満たすようにデザインされており，V と E により構成されている．V（E）は，c_{now} における頂点（辺）集合である．端的に言うと，ウィンドウ上のトランザクションに現れる o_i は，頂点 v_i と見なされ，パターンを表すために辺を作成する．以下に詳細を説明する．

辺. トランザクション $t=\langle j,O\rangle$ が与えられた時，以下の 2 つの要件を満たすように辺を作成する．

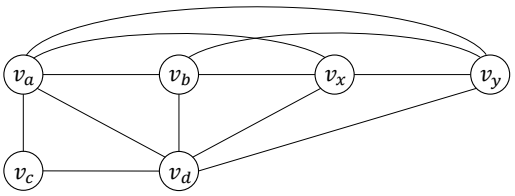
（1）O において，順に現れるオブジェクト間に辺が存在する．例えば，$O=\{o_p,o_q,o_r\}$ であれば，辺 $e_{p,q}$ および $e_{q,r}$ が作成される．

（2）O 中に排他的パターンが存在する場合，対応する頂点間に辺が存在する．例えば，$\{o_p,o_r\}\subset O$ が排他的パターンである場合，$e_{p,r}$ が作成される．この要件は，排他的パターンを直接辿り，排他的共起パターンを効率的に列挙するための重要な要件である．

各辺 e は，ラベルの順序付き集合（$e.L$）を持つ．今，c_{now} において $e_{p,q}$ が作成されると想定する．この時，ラベル $\langle P,j,c_{now}\rangle$ が $e_{p,q}.L$ に挿入される．$P'=\{\forall o_q\in O\,|\,o_p\prec o_q\}$ とすると，$P=O\backslash P'$ である．各ラベルは，作成時刻順（c）により管理される．以降，$e_{p,q}$ と記する場合は，$o_p\prec o_q$ とする．

頂点. トランザクション $t=\langle j,O\rangle$（$o_q\in O$）が c_{now} において生成された時，$v_q\notin V$ であれば v_q を作成する．各頂点 v_q は，タプル集合 $v_q.S$，フラグ $v_q.f$，および転置ファイル $v_q.I$ を持つ．O に含まれるオブジェクトに対応する頂点に対して，$\langle j,c_{now}\rangle$ がそのタプル集合に追加される．（$\langle j,c\rangle\in v_q.S$ であれば，c は c_{now} に置換される．）$v_q.S$ が c_{now} に更新されれば，$v_q.f=1$ であり，そうでなければ $v_q.f=0$ である．$\overrightarrow{\langle P,j,c\rangle}$ を $\langle P,j,c\rangle\in e_{p,q}.L$ に対するポインタとする．$v_q.I$ は，$\bigcup_E e_{p,q}.L$ に現れる全てのオブジェクトの集合およびポスティングリストにより構成される．各ポスティングリスト $v_q.I(o)$ は，$\overrightarrow{\langle P,j,c\rangle}$ の集合であり，$o\in P$ かつ $\langle P,j,c\rangle\in\bigcup_E e_{p,q}.L$ である．

例 3. 図 2 は，表 1 のトランザクションに基づく CP-Graph のグラフ構造を示している．そのトランザクション集合では，オブジェクト o_a，o_b，o_c，o_d，o_x，および o_y が現れるため，CP-Graph は頂点 v_a，v_b，v_c，v_d，v_x，および v_y を持つ．また，t_1 により，辺 $e_{a,b}$，$e_{b,x}$，および $e_{x,y}$ が生成される．同様に，排他的パターン（例えば $\{o_a,o_y\}$）も辺（$e_{a,y}$）が作成されている．次に，各辺のラベル集合に焦点を当てる．例えば，$\bigcup_E e_{i,y}.L$（$o_i\prec o_y$）に現れるオブジェクトの集合は，$\{o_a,o_b,o_c,o_d,o_x\}$ である．そのため，$v_y.I$ はポスティングリストとして，$v_y.I(o_a)$，$v_y.I(o_b)$，$v_y.I(o_c)$，$v_y.I(o_d)$，および $v_y.I(o_x)$ を持つ．（$v_y.I(o_a)$ によりポインタが管理されているラベルは，太字で示されている．）■



辺 e	ラベル集合 $e.L$
$e_{a,b}$	$\langle\{o_a\}, 1, c'\rangle, \langle\{o_a\}, 4, c'\rangle$
$e_{a,c}$	$\langle\{o_a\}, 3, c'\rangle$
$e_{a,d}$	$\langle\{o_a\}, 4, c'\rangle$
$e_{a,x}$	$\langle\{o_a\}, 3, c'\rangle$
$e_{a,y}$	$\langle\{\boldsymbol{o_a}\}, \boldsymbol{4}, \boldsymbol{c'}\rangle$
$e_{b,d}$	$\langle\{o_a, o_b\}, 4, c'\rangle$
$e_{b,x}$	$\langle\{o_a, o_b\}, 1, c'\rangle$
$e_{b,y}$	$\langle\{\boldsymbol{o_a}, \boldsymbol{o_b}\}, \boldsymbol{4}, \boldsymbol{c'}\rangle$
$e_{c,d}$	$\langle\{o_a, o_c\}, 3, c'\rangle$
$e_{d,x}$	$\langle\{o_d\}, 2, c'\rangle, \langle\{o_a, o_c, o_d\}, 3, c'\rangle$
$e_{d,y}$	$\langle\{\boldsymbol{o_a}, \boldsymbol{o_b}, \boldsymbol{o_d}\}, \boldsymbol{4}, \boldsymbol{c'}\rangle$
$e_{x,y}$	$\langle\{\boldsymbol{o_a}, \boldsymbol{o_b}, \boldsymbol{o_x}\}, \boldsymbol{1}, \boldsymbol{c'}\rangle, \langle\{o_d, o_x\}, 2, c'\rangle, \langle\{\boldsymbol{o_a}, \boldsymbol{o_c}, \boldsymbol{o_d}, \boldsymbol{o_x}\}, \boldsymbol{3}, \boldsymbol{c'}\rangle$

図 2　表 1 のトランザクションに基づく CP-Graph のグラフ構造と各辺のラベル集合

次に，CP-Graph を用いてあるパターンの μ およびその上界値をどのように計算するか説明する．パターン $\{o_p, o_q\}$ が与えられた時，$v_q.I$ を用いて $\mu(\{o_p, o_q\})$ の上界値は容易に得られる．

定理 1. 頂点 v_q について，$\mu(\{o_p, o_q\}) \leqq |v_q.I(o_p)|$ が成り立つ．

アプリオリ性質により，$|v_q.I(o_p)|$ は $\{o_p, o_q\}$ の上位集合の μ の上界値でもある．CP-Graph において $v.I$ をハッシュテーブルで実装することにより，$|v_q.I(o_p)|$ は $\mathcal{O}(1)$ で計算可能である．定理 1 を利用することにより，不必要なパターンの列挙を枝刈りできる．また，あるパターン P の $\mu(P)$ の計算も効率的に実行できる．

- $|P| = 1$ の場合．$P = \{o_p\}$ とすると，$|v_p.S| = \mu(P)$ であり，定数時間で求められる．
- $|P| = 2$ の場合．$P = \{o_p, o_q\}$ とすると，$v_q.I(o_p)$ に含まれる全てのストリームの識別子を S に追加し，$|S|$ を返す．これは $\Theta(|v_q.I(o_p)|)$ 時間で求められる．
- $|P| \geqq 3$ の場合．$P = \{o_a, o_p, ..., o_q\}$ とする．ラベルのアクセス回数を削減するために，

$$o^* = \underset{\forall e_{i,q} \in E, o_i \in P}{\operatorname{argmin}} |v_q.I(o_i)|.$$

を計算する．その後，$\forall \langle P', j, c\rangle \in v_q.I(o^*)$（$P \backslash \{o_q\} \subseteqq P'$ かつ $j = 1, 2, ..., |\mathcal{S}|$）に対して，$|P| = 2$ の場合と同様の処理を実行する．この処理の計算量は，$\Theta(n + |v_q.I(o^*)|)$ であり，$n = |\{\forall e_{i,q} \in E \,|\, o_i \in P\}|$ である．

4.2　CP-Graph の更新

本節では，CP-Graph をインクリメンタルに更新する手順について説明する．

4.2.1　新しいトランザクションの挿入

4.1 節で紹介した通り，辺を作成する際は，2 つの要件を満たす必要がある．ここで，1 つ目の要件を満たすのは容易である．トランザクション中のオブジェクト集合において，順に現れるオブジェクト間に辺を作成すれば良い．しかし，この作成方法だけでは 2 つ目の要件を満たせない．$\{o_p, o_r\}$ を排他的パターンとすると，o_p と o_r が順に現れるトランザクションが存在しない可能性がある．例えば，例 2 において $\{o_a, o_y\}$ は排他的パターンだが，表 1 のトランザクションにおいて，o_y が o_a の直後に現れることはない．

そこで，2 つ目の要件を満たし，かつ効率的に CP-Graph を更新するアプローチが必要となる．今，$t = \langle i, O\rangle$（$|O| \geqq 3$, $o_p, o_r \in O$, および o_p と o_r は O において順に現れない）が与えられたとする．もし $e_{p,r} \notin E$ であれば，$e_{p,r}$ を作成する必要があるか考える．具体的には，$\{o_p, o_r\}$ が排他的パターンである場合，$e_{p,r}$ を作成する必要がある．これを確かめるために，$\mu(\{o_p, o_r\})$ および $\max_{o_q \in O'} \mu(\{o_p, o_q, o_r\})$ を計算する．また，

$$O' = \{o_q \,|\, e_{q,r} \in E, \langle\{o_i, ..., o_p, ..., o_q\}, j, c\rangle \in e_{q,r}.L\} \quad (1)$$

である．（もし $\mu(\{o_p, o_r\}) = \max_{o_q \in O'} \mu(\{o_p, o_q, o_r\})$ であれば，$\{o_p, o_r\}$ は排他的パターンでない．）ここで，$\max_{o_q \in O'} \mu(\{o_p, o_q, o_r\})$ の計算は時間がかかる．なぜなら，$\forall o_q \in O'$ に対して，$v_r.I(o_i) = \operatorname{argmin}\{|v_r.I(o_p)|, |v_r.I(o_q)|\}$ をスキャンしなければならないからである．そのため，多少のフォルスポジティブを許容することにより，この計算コストを削減する．O' において識別子が最も小さいオブジェクトである $o_{q'}$ を特定し，$\mu(\{o_p, o_r\})$ および $\mu(\{o_p, o_{q'}, o_r\})$ を計算する．これらは，$v_r.I(o_p)$ のスキャンにより同時に計算可能である．その結果，$\mu(\{o_p, o_r\}) > \mu(\{o_p, o_{q'}, o_r\})$ であれば $e_{p,r}$ を作成する．（$\mu(\{o_p, o_{q'}, o_r\}) \leqq \max_{o_q \in O'} \mu(\{o_p, o_q, o_r\})$ が成り立つため，フォルスネガティブは起きない．）この処理を連鎖更新と呼ぶ．

挿入アルゴリズム． c_{now} において $t = \langle i, O\rangle$ が発生したとする．今，o_r を O における j 番目のオブジェクトとする．もし $v_r \notin V$ であれば，v_r を作成する．その後，$v_r.S$ を更新し，$v_r.f = 1$ とする．

ここで，$j \geqq 2$ とし，o_q を O における（$j-1$）番目のオブジェクトとする．$e_{q,r} \notin E$ であれば，$e_{q,r}$ を作成する．P を，O における 1 番目から（$j-1$）番目におけるオブジェクトの集合とすると，$e_{q,r}.L$ に $\langle P, i, c_{now}\rangle$ を追加する．そして，$\forall o_p \in P$ に対して，$\langle P, i, c_{now}\rangle$ のポインタを $v_r.I(o_p)$ に追加する．

さらに，$j \geqq 3$ である場合，$\exists e_{q',r} \in E$（$o_{q'} \neq o_q$）であるか確認する．$e_{q',r} \notin E$ である場合，$P = \{o_p, ..., o_r\}$（$|P| \geqq 3$）には必ず o_q が含まれており，$\{o_p, o_r\}$ は排他的パターンでないことが保証される．一方，$e_{q',r} \in E$ である場合，$\forall o_p \in P \backslash \{o_q\}$（$e_{p,r} \notin E$）に対して，連鎖更新を実行する．（連鎖更新によって辺が作成された場合も，前述の方法と同様にラベルが追加される．）その後，消去リスト L_{del} に $\langle c_{now}, v_r, o_p, 0\rangle$ を挿入する．

上の操作後，$j < |O|$ であれば，（$j+1$）番目のオブジェクトに対して上の操作を実行する．一方 $j = |O|$ であれば，L_{del} に $\langle c_{now}, v_r, o_q, 1\rangle$ を挿入し，t に関する更新を終了する．

4.2.2 ウィンドウから外れたトランザクションの消去

新たなトランザクションの挿入処理の際，トランザクションを要約した情報を L_{del} に挿入している．L_{del} に含まれている情報はトランザクションの全ての情報でないにも関わらず，ウィンドウから外れたトランザクションの消去を正確に実行できる．

性質 5. c_{now} における L_{del} の最初の要素を $\langle c_{now}-w, v_r, o_q, f\rangle$（$o_q \neq \varnothing$）とする．$e_{q,r}.L$ の最初のラベルの作成時刻は $c_{now}-w$ である．

このアプローチは，メモリ効率を上げるだけでなく，消去の必要があるラベルを $\mathcal{O}(1)$ で特定できるという利点がある．

削除アルゴリズム. $\langle c_{now}-w, v_r, o_q, f\rangle \in L_{del}$ が与えられた時，(i) $o_q = \varnothing$，(ii) $f = 0$，および (iii) $f = 1$ の 3 つの場合がある．場合 (i) の時，ウィンドウから外れるトランザクションは o_r のみ保持しているため，$v_r.S$ を更新し，$v_r.f = 1$ とすれば良い．場合 (ii) および (iii) の時，$e_{q,r}.L$ の最初のラベルを $\langle\{o_p, ..., o_q\}, i, c_{now}-w\rangle$ とする．$f = 0$ であれば，$e_{q,r}$ は連鎖更新により作成されているため，このラベルを削除すれば良い．また，$e_{q,r}.L = \varnothing$ となれば，$e_{q,r}$ も削除する．一方，$f = 1$ であれば，v_r から順にラベルに対応する辺を辿り，ラベル，辺，および頂点を更新する．($v.S$ および $e.L$ が $\varnothing$ であれば, CP-Graph から v および e を削除する.) その後，L_{del} から $\langle c_{now}-w, v_r, o_q, f\rangle$ を削除する．

4.3 閾値の計算

性質 4 より，CP-Graph の更新後，閾値を計算する必要がある．また，トランザクションの挿入および削除により，いくつかの共起パターンの μ は更新されている．CP-Graph は，$c_{now}-1$ から μ の更新が起きたパターンを容易に識別できる．$\exists o \in P$ に対して $v \notin V$ であれば，P はウィンドウ上に存在しないことが分かる．同様に，$v.f = 0$ であれば，P を含むトランザクションは生成も削除も行われていないことが分かる．そのため，以下の性質が成り立つ．

性質 6. μ' を $c_{now}-1$ における P の $\mu(P)$ とすると，

$$\mu(P) = \begin{cases} 0 & (\exists o \in P, v \notin V) \\ \mu' & (\exists o \in P, v.f = 0) \end{cases}$$

であり，$\forall o \in P$ に対して $v.f = 1$ であれば，$\mu(P)$ の値は不明である．

次に，μ の更新により，$c_{now}-1$ において解に含まれていたパターンが c_{now} において排他的共起パターンでなくなることがある．以下の性質により，そのようなパターンを特定できる．

性質 7. c_{now} において，全ての $\langle P, \mu(P)\rangle \in \mathcal{A}$ について $\mu(P)$ が正確な値であるとする．$\langle P, \mu(P)\rangle$ が与えられた時，(i) $\mu(P) = \mu(P')$ かつ $P \subset P'$ である $\langle P', \mu(P')\rangle$ が $\mathcal{A}$ に存在しないか，(ii) $P \not\subset Q$ かつ $\mu(P) = \mu(Q) = \mu(P \cup Q)$ となる $\langle Q, \mu(Q)\rangle$ が $\mathcal{A}$ に存在すれば，P は排他的共起パターンでないことが分かる．(ii) の場合，Q も排他的共起パターンではなく，$\forall o \in P \cup Q$ に対して $v.f = 1$ である．

以上の 2 つの性質を考慮し，$c_{now}-1$ における解から閾値を得ることを試みる（アルゴリズム 1）．まず，集合 $\mathcal{M}$（初期値 $\varnothing$）を準備する．次に，$\mathcal{A}$ の要素を $|P|$ の大きい順にソートする．その後，$\forall\langle P, \mu(P)\rangle \in \mathcal{A}$ に対して以下の操作を行う．まず，性質 6 から $\mu(P)$ を得る（4–8 行）．P が共起パターンでなくなれば，$\langle P, \mu(P)\rangle$ を $\mathcal{A}$ から取り除く（9–10 行）．P が共起パターンであれば，性質 7 から P が排他的共起パターンであるか計算する（12–23 行）．性質 7 における場合 (i) および (ii) のどちらにも該当しなければ，$\mu(P)$ を $\mathcal{M}$ に挿入する（25 行）．

Algorithm 1: $\mathcal{A}$-REVISITED($\mathcal{A}$)

Input: CP-Graph and $\mathcal{A}$ //$\mathcal{A}$ is the answer set at $c_{now}-1$
Output: $\mathcal{M}$ //$\mathcal{M}$ is a set of μ

```
1  M ← ∅
2  Sort ⟨P, μ(P)⟩ ∈ A in descending order of |P|
3  for ∀⟨P, μ(P)⟩ ∈ A do
4      if ∃o ∈ P such that v ∉ V then
5          μ(P) ← 0
6      else
7          if ∀o ∈ P, v.f = 1 then
8              μ(P) ← μ-COUNT(P)
9      if μ(P) ≦ 1 then
10         A ← A − {⟨P, μ(P)⟩}
11     else
12         Let P' be a co-occurrence pattern where ⟨P', μ(P')⟩ ∈ A and
           μ(P') has been computed
13         f ← 0
14         for ∀⟨P', μ(P')⟩ ∈ A where μ(P) = μ(P') do
15             if P ⊂ P' then
16                 A ← A − {⟨P, μ(P)⟩}
17                 f ← 1
18                 break
19             if ∀o ∈ P ∪ P', v.f = 1 then
20                 μ ← μ-COUNT(P ∪ P')
21                 if μ = μ(P) then
22                     f ← 1
23                     break
24         if f = 0 then
25             M ← M ∪ {μ(P)}
26 return M
```

$c_{now}-1$ における解に含まれるパターンが排他的共起パターンでなくなった場合，$|\mathcal{M}| < k$ である．この時，閾値は CP-Graph を辿ることによって計算する．CP-Graph は，与えられたパターンの μ を（$|P| \leqq 2$ の場合に特に）効率的に計算できるため，アルゴリズム 2 により，タイトな閾値を計算できる．

閾値の計算アルゴリズム. まず，$|\mathcal{M}| = k$ であれば，$\mathcal{M}$ における k 番目に大きい値を閾値とする（2 行）．$|\mathcal{M}| < k$ であれば，CP-Graph から閾値を求める．$\mathcal{M}$ における $|\mathcal{M}|$ 番目に大きい値を μ' とし，$\sigma = |\{\mu \in \mathcal{M} \,|\, \mu = \mu'\}|$ とする（4 行）．次に，$\forall v_j \in V$ について，$\mu(\{o_j\})$ を計算し，$2 \leqq \mu(\{o_j\}) \leqq \mu'$ であれば，

$$\mu^p(o_j) = \underset{\forall e_{i,j} \in E}{\operatorname{argmax}}\ \mu(\{o_i, o_j\}).$$

を計算する．$\mu^p(o_j)$ は，o_j を最後のオブジェクトとするパター

Algorithm 2: THRESHOLD-COMPUTATION($\mathcal{M}$)

Input: CP-Graph and $\mathcal{M}$
Output: threshold

1 **if** $|\mathcal{M}| = k$ **then**
2 return the kth highest μ among $\mathcal{M}$
3 $\mu' \leftarrow$ the $|\mathcal{M}|$th highest μ among $\mathcal{M}$
4 $\sigma \leftarrow |\{\mu \in \mathcal{M} \mid \mu = \mu'\}|$
5 **for** $\forall v_j \in V$ *where* $2 \leqq \mu(\{o_j\}) \leqq \mu'$ **do**
6 $\mu^p(o_j) \leftarrow 0$
7 **for** $\forall e_{i,j} \in E$ **do**
8 $\mu \leftarrow \mu$-COUNT($\{o_i, o_j\}$)
9 **if** $\mu > \mu^p(o_j)$ **then**
10 $\mu^p(o_j) \leftarrow \mu$
11 **if** $\mu(\{o_j\}) > \mu^p(o_j)$ **then**
12 $\mathcal{M} \leftarrow \mathcal{M} \cup \mu(\{o_j\})$
13 **if** $|\mathcal{M}| < k + \sigma$ **then**
14 return 2
15 return the $(k + \sigma)$th highest μ among $\mathcal{M}$

ン集合の中における最大値である．そのため，$\mu(\{o_j\}) > \mu^p(o_j)$ であれば，$\{o_j\}$ またはその上位集合 $\{o_j, ..., o_{j'}\}$ は排他的共起パターンであり，$\mu(\{o_j\})$ は $\mathcal{M}$ に追加される（5–12 行）．そして，$\mathcal{M}$ の中で（$k + \sigma$）番目の μ を閾値とする．k 番目としないのは，$\mathcal{M}$ には σ 個の μ' が存在するからである．（もし $|\mathcal{M}| < k + \sigma$ であれば，閾値は 2 とする．）以上の操作により，定理 2 が成立する．

定理 2. c_{now} における k 番目に大きい μ を τ_{now} とし，アルゴリズム 2 によって得られる閾値を τ とすると，$\tau_{now} \geqq \tau$ である．

4.4 解の更新

閾値の計算後，CP-Graph を辿ることにより $\mathcal{A}$ を更新する．アプリオリ性質と τ により，$|v.S| < \tau$ である頂点は全て枝刈りできる．つまり，$|v_i.S| \geqq \tau$ である v_i が与えられた時，共起パターン $P = \{o_i, ..., o_j\}$（$\mu(P) \geqq \tau$）を列挙し，$\mathcal{A}$ を更新する．以下の場合があればさらに探索空間を削減できる．

定理 3. 以下の 2 つの場合を考える．
（I）$\tau > \tau_{now-1}$.
（II）$\tau = \tau_{now-1}$ かつアルゴリズム 1 において $|\mathcal{M}| = k$ である．
アルゴリズム 2 の実行後に，場合（I）または（II）であった時，$\mathcal{A}$ に新しく P が追加されるならば，$\forall o \in P$ に対して $v.f = 1$ である．

ここから，$\mathcal{A}$ の更新アルゴリズムである $\mathcal{A}$-REFINEMENT（アルゴリズム 3）を紹介する．説明の簡易化のために，定理 3 における場合（I）または（II）の状態であることを想定し，説明の最後にいずれの場合でもない状態の時の差分について紹介する．

$|v_j.S| \geqq \tau$ かつ $v_j.f = 1$ である $v_j \in V$ が与えられたとする．
（1）まず $v_j.f = 0$ とし，キュー $\mathcal{Q}$ に $\langle o_j, \mu(\{o_j\})\rangle$ を挿入する．
（2）$\{o_j, o_q\}$ に焦点を当てる．このとき，$|v_q.I(o_j)| \geqq \tau$ かつ $\langle P', i, c\rangle \in e_{j,q}.L$ であり，$\{o_j, o_q\} \subseteqq P'$ である．（$|v_q.I(o_j)| < \tau$ となる $\{o_j, o_q\}$ は定理 1 により枝刈りできる．）もし $\mu(\{o_j, o_q\}) \geqq \tau$ かつ $v_q.f = 1$ であれば，バッファ $\mathcal{B}$ に $\langle\{o_j, o_q\}, \mu(\{o_j, o_q\})\rangle$ を挿入する．ここで，$\mu(o_j) > \max \mu(\{o_j, o_q\})$ であれば，$\{o_j\}$ は排他的共起パターンである可能性がある．
（3）次に，$\{o_j\}$ の上位集合が $\mathcal{A}$ に存在するか確認する．もし $\{o_j\} \subseteqq P'$ かつ $\mu(\{o_j\}) = \mu(P')$ となる $\langle P', \mu(P')\rangle$ が $\mathcal{A}$ に存在しなければ，$\mathcal{A}$ における k 番目の要素と $\langle\{o_j\}, \mu(\{o_j\})\rangle$ を置換し，τ を更新する．
（4）その後，$\mathcal{Q}$ の先頭の要素をポップする．$\mathcal{B}$ の各要素（$\langle\{o_j, o_q\}, \mu(\{o_j, o_q\})\rangle$）が $\mu(\{o_j, o_q\}) \geqq \tau$ であれば，$\mathcal{Q}$ に

Algorithm 3: $\mathcal{A}$-REFINEMENT(τ, τ_{now-1})

Input: CP-Graph, τ, τ_{now-1} //τ is the threshold obtained by Algorithm 2
Output: $\mathcal{A}$

1 $i \leftarrow 1$
2 **if** $\tau > \tau_{now-1}$ **then**
3 $i \leftarrow 0$
4 **else**
5 **if** $\tau = \tau_{now-1}$ *and Algorithm 1 returns* $\mathcal{M}$ *of* $|\mathcal{M}| = k$ **then**
6 $i \leftarrow 0$
7 **case** $i = 0$
8 **for** $\forall v_j \in V$ *where* $|v_j.S| \geqq \tau$ *and* $v_j.f = 1$ **do**
9 $v_j.f = 0$
10 $\mathcal{Q} \leftarrow \{\langle\{o_j\}, \mu(\{o_i\})\rangle\}$
11 **while** $\mathcal{Q} \neq \varnothing$ **do**
12 $\langle P, \mu(P)\rangle \leftarrow \mathcal{Q}$.front //let $P = \{o_a ..., o_q\}$
13 $\mu \leftarrow 0, \mathcal{B} \leftarrow \varnothing$
14 **for** $\forall e_{q,r} \in E$ *where* $|v_r.I(o_q)| \geqq \tau$ **do**
15 **if** $\exists\langle P', i', c\rangle \in e_{q,r}.L$ *such that* $P \subseteqq P'$ **then**
16 $\mu' \leftarrow \mu$-COUNT($P \cup \{o_r\}$)
17 **if** $\mu' \geqq \tau$ **then**
18 **if** $v_r.f = 1$ **then**
19 $\mathcal{B} \leftarrow \mathcal{B} \cup \langle P \cup \{o_r\}, \mu(P \cup \{o_r\})\rangle$
20 **if** $\mu' > \mu$ **then**
21 $\mu \leftarrow \mu'$
22 **if** $\mu(P) > \mu$ **then**
23 **if** $\nexists\langle P', \mu(P')\rangle \in \mathcal{A}$ *such that* $P \subseteqq P'$ *and* $\mu(P) = \mu(P')$ **then**
24 Let P_k be the co-occurrence pattern with the kth highest μ
25 $\mathcal{A} \leftarrow \mathcal{A} \cup \langle P, \mu(P)\rangle - \langle P_k, \mu(P_k)\rangle$
26 **else**
27 **if** $\exists\langle P', \mu(P')\rangle \in \mathcal{A}$ *such that* $P' \subseteqq P$ *and* $\mu(P) = \mu(P')$ **then**
28 $\mathcal{A} \leftarrow \mathcal{A} - \{\forall\langle P', \mu(P')\rangle \in \mathcal{A} \mid P' \subseteqq P, \mu(P) = \mu(P')\}$
29 $\mathcal{A} \leftarrow \mathcal{A} \cup \langle P, \mu(P)\rangle$
30 **if** $|\mathcal{A}| = k$ **then**
31 $\tau \leftarrow$ the kth highest μ in $\mathcal{A}$
32 $\mathcal{Q} \leftarrow \mathcal{Q} - \langle P, \mu(P)\rangle$ //pop $\mathcal{Q}$.front
33 $\mathcal{Q} \leftarrow \{\forall\langle P', \mu(P')\rangle \in \mathcal{Q} \mid \mu(P') \geqq \tau\} \cup \{\forall\langle P', \mu(P')\rangle \in \mathcal{B} \mid \mu(P') \geqq \tau\}$
34 **case** $i = 1$
35 Execute the same operation as case $i = 0$ without the condition of $v.f = 1$
36 return $\mathcal{A}$

挿入する．

もし $\mathcal{Q} = \varnothing$ であれば，$|v.S| \geqq \tau$ となる別の頂点に対して（1）–（4）の操作を実行する．そうでなければ，$\mathcal{Q}$ の先頭の要素（$\langle\{o_j, o_q\}, \mu(\{o_j, o_q\})\rangle$）に対して，$\mu = \max_{\forall e_{q,r} \in E} \mu(\{o_j, o_q, o_r\})$ を計算し，操作（2）と同様にバッファを更新する．ここで，$e_{q,r}.L$ は $\{o_j, o_q, o_r\}$ を持つラベルを保持している．もし $\mu(\{o_j, o_q\}) > \mu$ であれば，操作（3）と同様の場合について考える．$P' \subseteqq \{o_j, o_q\}$ かつ $\mu(\{o_j, o_q\}) = \mu(P')$ となる P' について $\langle P', \mu(P')\rangle \in \mathcal{A}$ であれば，$\mathcal{A}$ から $\langle P', \mu(P')\rangle$ を取り除き，$\langle\{o_j, o_q\}, \mu(\{o_j, o_q\})\rangle$ を加える．その後，操作（4）を実行し，$\mathcal{Q}$ をチェックする．上記の操作を $|v.S| \geqq \tau$ となる全ての $v \in V$ について実行する．最後に，定理 3 における場合（I）および（II）の状態でなければ，“$v.f = 1$” という条件を除外する．

5. 評価実験

本章では，提案アルゴリズムの性能評価のために行った実験の結果を紹介する．

5.1 セッティング

本実験では以下の 2 つのアルゴリズムを評価した．

- [12] における提案アルゴリズム（Seg-tree と表記する）.（これは，既存研究において，複数ストリームから共起パターンを正確にマイニングできる唯一のアルゴリズムである.）3. 章で述べた通り，CooMine は動的な閾値に対応していないため，新たな（ウィンドウから外れた）トランザクションから閾値を推定するアプローチを CooMine に組み込んだ．
- 本稿の提案アルゴリズム（CP-Graph と表記する）.

上の 2 つのアルゴリズムは C++で実装されており，すべての実験は 3.4GHz Intel Core i7 および 32GB RAM で構成される PC 上で行った．

評価指標. 本実験では，以下の 3 つの指標を評価した．

- インデックス更新時間：ウィンドウの各スライドにおけるインデックスの平均更新時間
- 解を更新するまでの時間：ウィンドウの各スライドにおける平均処理時間（インデックスの更新時間を含む.）
- メモリ使用量：インデックスによるメモリの平均消費量

データセット. 本実験では，2 つの実データ OnlineRetail(注2) および Twitter を用いた．OnlineRetail は 2010/01/12 から 2011/09/12 間で得られたオンライン購買に関するトランザクションの集合である．各トランザクションに，[0,9999] から選ばれたランダムな整数値をストリームの識別子として与えた．Twitter は，2015/4/3 から 2015/4/9 で得られた英語ツイートの集合である．一つのツイート中に現れる単語をオブジェクトとすることにより，1 つのツイートを 1 つのトランザクションと見なす [12]．1 人のユーザを 1 つのストリームと見なすと，ストリームの数は 3,341,975 である．

（注2）：https://archive.ics.uci.edu/ml/datasets

トランザクションの集合を T，T に含まれるオブジェクトの集合を D とする．表 2 は，それぞれのデータ集合の統計値を表している．

表 2　データセットの統計値

データセット	$\|T\|$	T における $\|D\|$ の平均	$\|D\|$
OnlineRetail	541,909	4.36	2,603
Twitter	5,677,156	4.06	477,990

5.2 実験結果

本節では，k および w を変えて行った実験の結果を示す．スライディングウィンドウモデルとして，OnlineRetail は数に基づくウィンドウ，Twitter では時間に基づくウィンドウを用いた．OnlineRetail を用いた実験におけるデフォルト設定は，$k = 50$，$\Delta = 50$，および $w = 200,000$ である．一方，Twitter を用いた実験におけるデフォルト設定は，$k = 50$ および $w = 7,200$[sec] である．

k の影響. まず，表 3 はインデックス更新時間を示している．CP-Graph（および Seg-tree）の構造は k に依存しないため，インデックス更新時間（およびメモリ使用量）は k の影響を受けない．ここで，Seg-tree の方が CP-Graph よりもインデックス更新時間が短いことが分かる．トランザクション $t = \langle i, O\rangle$ が与えられた時，Seg-tree は O 中のオブジェクトに対応する頂点を作成し，順に現れる頂点間に辺を作成する．一方，CP-Graph は $|O| \geqq 3$ の場合に連作更新を行う．また，CP-Graph はトランザクションの削除の際に，転置ファイルも更新しなければならない．そのため，CP-Graph のインデックス更新時間が Seg-tree に比べて長くなる．

表 3　インデックス更新時間 [msec]

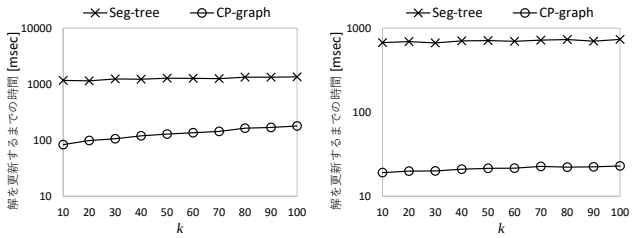
	OnlineRetail	Twitter
Seg-tree	0.15	2.12
CP-Graph	73.3	14.54

表 4 はメモリ使用量を示しており，CP-Graph のメモリ使用量は Seg-tree よりも大きいことが分かる．これは，$e.L$ および $v.I$ を管理しているためである．しかし，CP-Graph のメモリ使用量はメインメモリに容易にフィットする程度である．

表 4　メモリ使用量 [MB]

	OnlineRetail	Twitter
Seg-tree	38.49	32.90
CP-Graph	60.19	44.03

図 3 は，解を更新するまでの時間の結果を示している．まず，Seg-tree の処理時間は，$\mathcal{A}$ を更新する時間が大部分を占めていることが分かる．一方，CP-Graph の処理時間はインデックス更新時間が大部分を占めていることが分かる（表 3 を参照）．また，CP-Graph の性能は Seg-tree を大きく上回る．CP-Graph はインデックスの更新時間がやや長いものの，与えられたパターンの μ の計算や閾値の計算を効率的に実行できるため，$\mathcal{A}$ を更新する時間が早い．Seg-tree はインデックスは高速に更新でき



(a) 解を更新するまでの時間（OnlineRetail）　(b) 解を更新するまでの時間（Twitter）

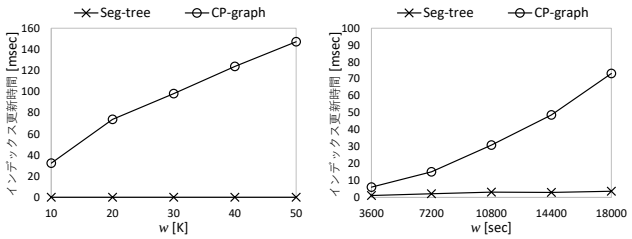
図 3　k の影響

るが，パターンの μ を計算する時間が長く，そのパターンが排他的であるかどうかを確認する時間も長い．

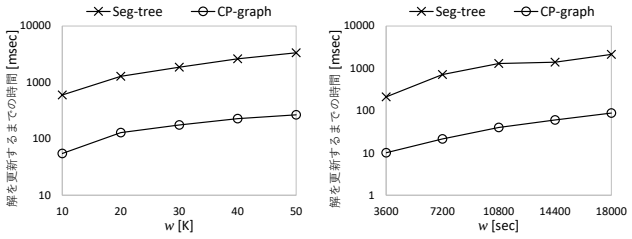
w の影響．次に，ウィンドウサイズの影響の結果（図 4）について紹介する．図 4(a)–4(b) から分かるように，ウィンドウサイズが大きくなると Seg-tree と CP-Graph は共にインデックス更新時間が長くなる．CP-Graph における連鎖更新では，複数の頂点，辺，およびラベルをチェックする必要がある．ウィンドウサイズが大きいと，それらの数も大きくなるため，連鎖更新にかかる時間も長くなる．

解を更新するまでの時間に焦点を当てる．図 4(c)–4(d) から，k の影響と同様に，CP-Graph の処理はインデックスの更新時間が大部分である．また，CP-Graph の処理時間は Seg-tree よりも早く，性能差はウィンドウサイズが大きくなるにつれて顕著になる．CP-Graph はあるパターン，例えば $\{o_i, ..., o_j\}$ の μ を，$v_j.I$ を用いるだけで計算できる．そのため，ウィンドウサイズに対してロバストであり，この結果は閾値の計算が効率的であり，タイトな閾値を与えていることを示している．

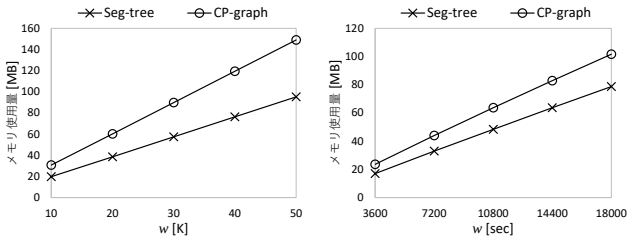
図 4(e)–4(f) はメモリ使用量を表している．Seg-tree および CP-Graph のメモリ使用量はウィンドウサイズに対して線形に増加している．CP-Graph では，ウィンドウサイズが大きくなると，辺が管理するラベル集合や頂点が管理する転置ファイルのサイズが大きくなる．そのため，Seg-tree よりも CP-Graph の方がメモリ使用量が大きい．



(a) インデックス更新時間（OnlineRetail）　(b) インデックス更新時間（Twitter）



(c) 解を更新するまでの時間（OnlineRetail）　(d) 解を更新するまでの時間（Twitter）



(e) メモリ使用量（OnlineRetail）　(f) メモリ使用量（Twitter）

図 4　w の影響

6. おわりに

近年の実世界アプリケーションにおいて，ストリームデータに対するパターンマイニングに多くの関心が集まっている．本稿では，複数のストリームが存在する環境において，最も多くのストリームに現れるパターンの上位 k 個のものをマイニングする問題に対する効率的なアルゴリズムを提案した．筆者らの知る限り，本問題はこれまでに着手されていない．本稿で提案した CP-Graph は，グラフと転置ファイルを組み合わせたデータ構造であり，与えられたパターンの μ を効率的に計算できる．また，CP-Graph は閾値の推定も効率的に実行でき，それにより解の更新を高速に行う．実データを用いた実験の結果から，提案アルゴリズムの有効性およびスケーラビリティを確認した．

謝辞．本研究の一部は，文部科学省科学研究費補助金・基盤研究 (A)(JP26240013)，若手 (B)(JP16K16056)，および JST 国際科学技術共同研究推進事業（戦略的国際共同研究プログラム）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] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, pages 487–499, 1994.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.

[3] Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz. Moment: Maintaining closed frequent itemsets over a stream sliding window. In *ICDM*, pages 59–66, 2004.

[4] J. Guo, P. Zhang, J. Tan, and L. Guo. Mining frequent patterns across multiple data streams. In *CIKM*, pages 2325–2328, 2011.

[5] R. Gwadera and F. Crestani. Discovering significant patterns in multi-stream sequences. In *ICDM*, pages 827–832, 2008.

[6] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, pages 1–12, 2000.

[7] B. Mozafari, H. Thakkar, and C. Zaniolo. Verifying and mining frequent patterns from large windows over data streams. In *ICDE*, pages 179–188, 2008.

[8] A. Salam and M. S. H. Khayal. Mining top-k frequent patterns without minimum support threshold. *KIS*, 30(1):57–86, 2012.

[9] L. Troiano and G. Scibelli. Mining frequent itemsets in data streams within a time horizon. *DKE*, 89:21–37, 2014.

[10] V. S. Tseng, C.-W. Wu, P. Fournier-Viger, and P. S. Yu. Efficient algorithms for mining the concise and lossless representation of high utility itemsets. *TKDE*, 27(3):726–739, 2015.

[11] J. Wang, J. Han, Y. Lu, and P. Tzvetkov. Tfp: An efficient algorithm for mining top-k frequent closed itemsets. *TKDE*, 17(5):652–663, 2005.

[12] Z. Yu, X. Yu, Y. Liu, W. Li, and J. Pei. Mining frequent co-occurrence patterns across multiple data streams. In *EDBT*, pages 73–84, 2015.

[13] X. Zhu and X. Wu. Discovering relational patterns across multiple databases. In *ICDE*, pages 726–735, 2007.