スライディングウィンドウ上での 位置・キーワード・ソーシャル関係に基づく Top-k パブリッシュ/サブスクライブ

西尾　俊哉† 　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: †{nishio.syunya,amagata.daichi,hara}@ist.osaka-u.ac.jp

あらまし　近年，多くのアプリケーションでは，PoI（Point of Interest）がパブリッシュ/サブスクライブ（Pub/Sub）モデルに基づいてデータを発信しており，ユーザは生成されたデータの中から自身が興味を持つもののみを取得する．また，位置情報サービスやソーシャルネットワークサービスの普及により，位置やキーワード，ソーシャル関係を用いた検索への関心が高まっている．本研究では，Pub/Sub モデルで生成されたデータから，ユーザにとって有用な上位 k 個のデータ（Top-k データ）をモニタリングする問題に取り組む．この上位 k 個のデータを収集する際，PoI の位置，指定したキーワードとの一致度，およびデータを生成した PoI とのソーシャル関係からデータのスコアを計算する．さらに，スライディングウィンドウモデルを適用することにより，最新のデータ集合における解をユーザに提供する．ウィンドウがスライドした際に，全てのクエリに対して Top-k データの更新をチェックする方法は，多数のクエリが存在する環境に対応できない．この問題を解決するため，クエリを四分木により管理し，生成されたデータが上位 k 個となり得るクエリにのみアクセスするアルゴリズムを提案する．さらに，k-スカイバンドを適用し，上位 k 個に含まれるデータが削除された場合においても，高速に Top-k データを更新する．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を示す．

キーワード　Pub/Sub，ソーシャルネットワーク，Top-k クエリ，スライディングウィンドウ

1. はじめに

スマートフォンやタブレットなどの GPS を利用可能な端末の普及に伴い，位置情報やキーワードに基づく検索が多くのアプリケーションにとって必要不可欠になっている [6], [9]．これらのアプリケーションの多くは，PoI からデータが与えられたとき，事前に登録された興味に基づいてユーザにデータを配信するパブリッシュ/サブスクライブ（Pub/Sub）モデルに基づいている [2], [4]．この Pub/Sub モデルでは，データの生成は頻繁に行われるため，大量のデータの中から，ユーザの趣向に応じてデータの順位を決定し，ユーザにとって有用な上位 k 個のデータを検索する Top-k 検索が実用的である [2]．また，Facebook(注1) や Twitter に代表される SNS の急速な発展により，ソーシャル関係を考慮した検索も注目を集めている [1], [7]．具体的には，SNS におけるソーシャル関係を用いたソーシャルフィルタリングによりユーザの嗜好を抽出し，ユーザの要求に合う検索が実現できる [3]．例えば，Facebook における “いいね” など，ユーザは興味のある PoI（Point of Interest）に対してソーシャル関係を持つ．この関係は，図 1 におけるユーザと PoI 間の辺として表現でき，図 1 はユーザ u_1 および u_2 のソーシャル関係を表現している．このとき，ユーザ u_2 は PoI p_1 とソーシャル関係があるため，p_1 が発信するデータを受信する．

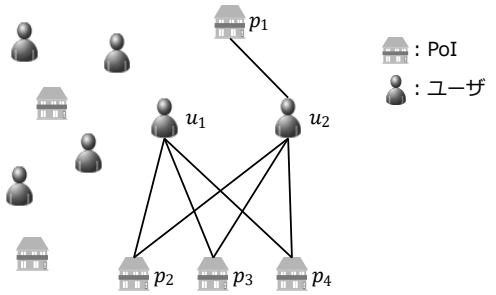


図 1: ユーザと PoI とのソーシャル関係

また，ユーザ u_1 と u_2 は，ソーシャル関係がある PoI が類似しているため，二人の嗜好は類似しているといえる．このとき，p_1 は u_1 とソーシャル関係はないが，p_1 は u_2 とソーシャル関係があるため，u_1 の嗜好に合う可能性は高い．そこで，ソーシャル関係を考慮し，p_1 が発信するデータを u_1 が受信しやすくすることにより，u_1 は既知でなかった嗜好に合うデータを受信できる．つまり，ソーシャル関係を考慮することにより，あるユーザにとって既知でない嗜好に合うデータも検索可能になると同時に，PoI は広告配信などを効率化できる．

本稿では，Pub/Sub モデルで生成されたデータから，ユーザにとって有用な上位 k 個のデータ（Top-k データ）をモニタリングする問題に取り組む．この Top-k クエリでは，PoI の位置，ユーザが指定したキーワードとの一致度，およびデータを

(注1)：https://www.facebook.com/

生成した PoI とのソーシャル関係からデータのスコアを計算する．さらに，最新のデータをユーザに提供するため，スライディングウィンドウモデル [5] を適用する．システムには多数の Top-k クエリが登録されており，Top-k データはクエリごとに異なる．そのため，クエリが多く存在している場合，ウィンドウがスライド（データの生成および削除）した際に，全てのクエリに対して Top-k データの更新をチェックする単純な方法は，多大な時間がかかってしまう．その結果，ウィンドウのスライドに対して，各ユーザの Top-k データを高速に更新できず，リアルタイム性を保証することが難しい．

この問題を解決するため，クエリを四分木 [8] により管理し，生成されたデータが Top-k データとなり得るクエリにのみアクセスするアルゴリズムを提案する．ノードに保存された情報から，生成されたデータが Top-k データとなり得ない部分木を枝刈りし，アクセスするクエリの数を削減する．さらに，各クエリに対して，後に Top-k データとなり得るデータ集合を保存しておく．上位 k 個に含まれるデータが削除されたとき，そのデータ集合を用いて Top-k データを更新し，不要なデータのアクセス数を削減する．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を確認した．本稿は，文献 [10] において筆者らが取り組んだ問題に対して，スライディングウィンドウモデルを適用した拡張版である．

以下では，2 章で本稿の問題を定義する．3 章で提案アルゴリズムについて説明し，4 章で実データを用いた実験の結果を示す．5 章で関連研究について述べ，最後に 6 章で本稿のまとめについて述べる．

2. 予 備 知 識

PoI の集合を P，全ての PoI が生成する全てのデータの集合を O，発行された全てのクエリの集合を Q とする．PoI から生成されたデータのうち，最新のデータをユーザに提供するため，スライディングウィンドウ W を用いる．本研究ではカウントベースのスライディングウィンドウを用いる．

定義 2.1 (スライディングウィンドウ)．大きさが $|W|$ のカウントベースのスライディングウィンドウには，直近に生成された $|W|$ 個のデータが含まれる．

データモデル． ある PoI p が生成するデータ $o \in O$ は，o の id（$o.id$），o を生成した PoI の id（$o.pid$），データの位置情報（$o.loc$），キーワード集合（$o.key$），および生成時間（$o.t$）を保持している．$o.loc$ は o を生成した PoI の位置とし，PoI の位置情報は緯度と経度によって表される 2 次元平面上の点とする．データのキーワード集合は，そのデータを表現（説明）するキーワードの集合（メタデータの一種）であり，更新はないものとする [2], [5], [6].

クエリモデル． あるユーザ u が発行するクエリ $q_u \in Q$ は，q_u の id（$q_u.id$），u が指定する位置情報（$q_u.loc$），1 つ以上の任意の数のキーワードの集合（$q_u.key$），および上位何個のデータを要求するかを指定する変数（$q_u.k$）を保持している．

各クエリの Top-k データを決定するため，クエリ q_u に対するデータ o のスコアを計算し，スコアの最も良い k 個のデータを Top-k データとする．このスコアは，距離スコア，キーワードスコア，およびソーシャルスコアに基づいて計算される．

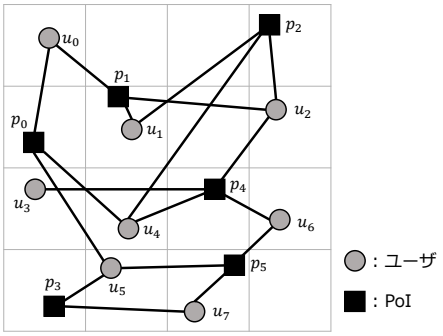


図 2: ソーシャルグラフ

定義 2.2 (距離スコア $dist(q_u, o)$)．$dist(q_u, o)$ は，式（1）により定義される．

$$dist(q_u, o) = 1 - \frac{d(q_u.loc, o.loc)}{MAXloc} \tag{1}$$

$d(q_u.loc, o.loc)$ は，クエリとデータとのユークリッド距離，$MAXloc$ はユーザと PoI が存在しうる領域の最大距離であり，これによりスコアを $[0, 1]$ の値に正規化している．その値を 1 から引くことにより，距離が近いものほどスコアが大きくなる．

定義 2.3 (キーワードスコア $key(q_u, o)$)．$key(q_u, o)$ は，式（2）により定義される．

$$key(q_u, o) = \frac{2|q_u.key \cap o.key|}{|q_u.key| + |o.key|} \tag{2}$$

このスコアは，クエリのキーワードとデータのキーワードの一致度を F 値を用いて計算したものである．

例 2.1. $q_u.key = \{w_1, w_2\}$ で，$o.key = \{w_2, w_3, w_4\}$ であるとき，$key(q_u, o) = \frac{2\times1}{2+3} = 0.4$ となる．

ソーシャルスコア $socio(q_u, p)$ は，クエリを発行したユーザとデータ o を生成した PoI p とのソーシャル関係により計算される．本稿で想定する環境では，ユーザは，Facebook における “いいね” のように，興味のある PoI に対してソーシャル関係を持つ．この関係は，ソーシャルグラフを用いて図 2 のように表現できる．丸がユーザ，四角が PoI を表し，ソーシャル関係はグラフにおける辺として表現できる．ここで，ソーシャルグラフの辺の集合を E とする．また，$e_{u,p}$ をユーザ u と PoI p 間の辺とする．

定義 2.4 (ソーシャルスコア $socio(q_u, p)$)．$socio(q_u, p)$ は，式（3）により定義される．

$$socio(q_u, p) = \begin{cases} 1 & (e_{u,p} \in E) \\ \max \frac{2|P_u \cap P_{u'}|}{|P_u| + |P_{u'}|} & (e_{u,p} \notin E) \end{cases} \tag{3}$$

ここで，$P_u = \{\forall p' \in P | \exists e_{u,p'} \in E\}$ であり，$P_{u'} = \{\forall p' \in P | (\exists e_{u',p} \in E) \wedge (\exists e_{u',p'} \in E)\}$ である．

つまり，$e_{u,p} \notin E$ である場合，ソーシャルスコアは，p について，$e_{u',p} \in E$ であるユーザ u' と u との PoI 間の辺の一致度を F 値を用いて計算した値の最大値である．

例 2.2. 例えば，図 2 において p_0 が o を生成した場合，$e_{u_0,p_0} \in E$ であるため，$socio(q_{u_0}, p_0) = 1$ である．次に，p_4 が o を生成した場合の $socio(q_{u_1}, p_4)$ について考える．ここで，$e_{u_1,p_4} \notin E$ である．p_4 との辺が存在するユーザは u_2，u_3，u_4，および u_6 であるため，それぞれのユーザとの PoI 間の辺の一致度を計算する．u_1 との辺が存在する PoI は p_1 および p_2 であり，u_2 との辺が存在する PoI は p_1，p_2，および p_4 であるため，u_1 と u_2 との PoI 間の辺の一致度は，$\frac{2\times2}{2+3} = 0.8$ となる．同様にして，u_3，u_4，および u_6 についても一致度を計算すると，u_1 と u_3 との一致度は 0，u_1 と u_4 との一致度は 0.4，また，u_1 と u_6 との一致度は 0 となる．よって，$socio(q_{u_1}, p_4) = \max\{0.8, 0, 0.4, 0\} = 0.8$ となる．

ここで，実際の環境ではソーシャル関係の更新（辺の追加および削除）が発生することが考えられる．この更新が起きると $socio(q_u, p)$ が変化する可能性がある．しかし，本研究ではソーシャル関係の更新は考えないものとし，更新が起きた場合においても既に計算済みの $socio(q_u, p)$ の値を利用するという方針をとる．

定義 2.5 (スコア関数)．クエリ q_u に対するデータ o のスコア $s(q_u, o)$ は，式（4）により定義される．

$$s(q_u, o) = dist(q_u, o) + key(q_u, o) + socio(q_u, p) \qquad (4)$$

スコアは大きいほど優れているものとする．

問題定義．多数のクエリおよびデータが与えられたとき，W 内のデータに対して，各クエリの Top-k データをリアルタイムにモニタリングする．

3. 提案アルゴリズム

データが新たに生成されたとき，位置およびキーワードに基づくスコアの計算は生成されたデータに依存し，異なる PoI から生成されたデータであっても，PoI 同士の位置が近く，似たキーワード集合を持つデータであれば，位置およびキーワードに関するスコアは近い値となる．また，ソーシャルスコアはユーザと PoI の関係のみに依存し，同じ PoI から生成されたデータに対するソーシャルスコアはデータごとに変化しない．

そこで，クエリの位置情報に基づいてクエリを四分木で管理し，各ノードにはそのノードに含まれるクエリの id，キーワード集合，および j（$j \geqq k$）番目のデータのスコアを集約した情報を保存する．四分木の構築の際，分割が終了したノードに対して，ノードの id，そのノードに含まれるクエリの id，および根ノードからそのノードまでの経路を表す数字列をノードリスト（L_N）に保存する．四分木を構築した後，各 PoI p_i に対応するソーシャルスコアリスト（L^i_{ss}）を作成する．L^i_{ss} は，L_N に含まれるノードごとに，ノード id およびそのノードに含まれるクエリのソーシャルスコアの最大値を保存したものである．クエリは事前にシステムに登録されているものとし [2], [6]，四分木の構築およびソーシャルスコアリストの作成はオフラインで行われる．

データが生成されると，そのデータのスコアが j 番目のデータのスコア以上となる可能性があるクエリにのみアクセスし，k-スカイバンドを更新するためのチェックを行う．まず，データを生成した PoI の L_{ss} と L_N を組み合わせて，チェックするノードのソーシャルスコアを得るために用いるソーシャルスコアテーブル T_{ss} を作成する．そして，四分木の根ノードから順に，ノードに保存された情報，データの位置情報とキーワード集合，および T_{ss} から，スコアの上界値を計算する．この上界値を用いて，k-スカイバンドの更新が不要なクエリへのアクセスを削減する．また，あるクエリの Top-k データに含まれるデータが削除されたとき，そのクエリの k-スカイバンドを用いて Top-k データを更新する．

まず，3.1 節において，提案アルゴリズムで適用する k-スカイバンドを説明する．次に，3.2 節において，クエリを管理する四分木の構築方法を紹介し，3.3 節において，各 PoI に対応する L_{ss} の作成方法を紹介する．最後に，3.4 節において，ウィンドウのスライドに対して，各クエリの Top-k データを更新するアルゴリズムを紹介する．

3.1 k-スカイバンド

ウィンドウのスライドにより，あるクエリの Top-k データに含まれるデータが削除されたとき，Top-k データを高速に更新するため k-スカイバンドを用いる．以下に，支配および k-スカイバンドを定義する．

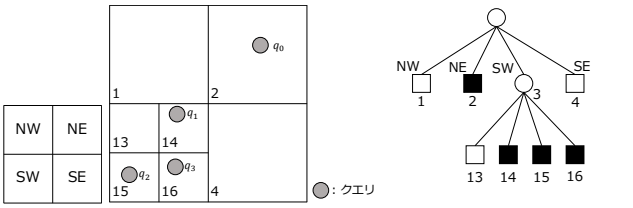
定義 3.1 (支配)．あるクエリ q_u に対して，データ o_1 がデータ o_2 に支配されるとき，$s(q_u, o_1) \leqq s(q_u, o_2)$ かつ $o_1.t < o_2.t$ を満たす．

定義 3.2 (k-スカイバンド)．あるクエリに対する k-スカイバンドは，W 内のデータにおいて，支配されるデータが k より少ないデータの集合である．

あるクエリに対する k-スカイバンドとは，W 内のデータのうち，古いデータが削除されたとき，Top-k データとなり得るデータの集合である．また，あるクエリにおける Top-k データは，k-スカイバンドに含まれるデータのうちスコアの大きい上位 k 個のデータである [5]．提案アルゴリズムでは，スコアが j（$j \geqq k$）番目のデータのスコアより大きくなる可能性のあるデータでのみ，k-スカイバンドの更新を行う．以降，提案アルゴリズムにおけるクエリ q_u の Top-k データを $q_u.D$，k-スカイバンドを $q_u.S$ とする．また，$q_u.S$ に含まれるデータ o において，o を支配するデータの数を $o.dom$ と表す．

3.2 四分木の構築

提案アルゴリズムでは，クエリを線形四分木を用いて管理する．四分木は，各ノードが 4 個までの子ノードを持つ木構造である．四分木のノードの分割を図 3 のように表現する．図 3a に示すように，ノードを分割した際，分割されたノードをそれぞれ，NW，NE，SW，および SE と名付ける．例えば，図 3b のように，4 つのクエリが存在し，領域が分割されたとする．



(a) (b) (c)

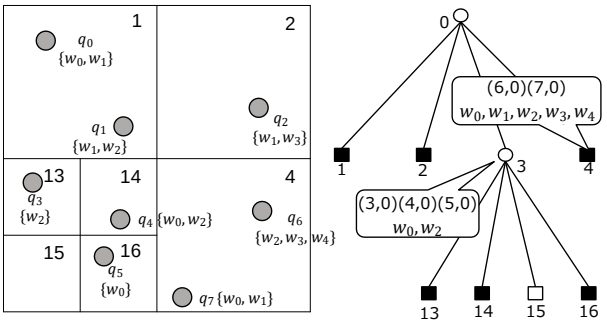
図 3: 四分木の表現方法

この場合，四分木の構造は，図 3c のような木構造をとる．図 3c に示すように，本稿では，葉ノードと中継ノードを表現するために四角と丸を用い，クエリが含まれる葉ノードは黒四角とする．

ここから，提案アルゴリズムにおける四分木の構築方法を紹介する．四分木を構築する際，クエリがどのノードに含まれるかはクエリの位置情報のみに依存する．まず，全ての PoI が存在している領域を根ノードとする．クエリは，全ての PoI が存在している領域内の任意の点を検索点とできるため，全てのクエリが根ノードに含まれる．次に変数 m を用意し，ノードに含まれるクエリの数が m 個を超える場合は，そのノードを分割し，クエリの位置情報に基づいて各ノードにクエリを振り分ける．分割されたノード内のクエリの数が m 個を超えていれば同じ操作を行い，あるノードに含まれるクエリの数が m 個以下になるまで繰り返す．図 2 におけるユーザが，自身の現在地を検索点とし，何らかのキーワードを指定したクエリ q_u を発行したと仮定する．$m = 2$ の場合，図 4a に示す四分木が構築される．

各ノードには，そのノードに含まれるクエリの id，キーワードの集合，およびクエリが管理している j 番目のデータのスコアを保存する．ここで，クエリ q_u が管理している j 番目のデータのスコアを $q_u.score_j$ と表し，その初期値は 0 とする．データの生成により $q_u.score_j$ が更新されると，そのクエリの情報を持つ全てのノードの情報が更新される．図 4a では，クエリの id およびそのクエリの $q_u.score_j$ を（id，$q_u.score_j$）と表している．ノード 4 には，クエリ q_6 および q_7 が含まれているため，それらの id，j 番目のデータのスコア，およびキーワードの集合が保存されている．ノード 3 には，自身の子ノードであるノード 13，14，15，および 16 に含まれるクエリ q_3，q_4，および q_5 の id，j 番目のデータのスコア，およびキーワードの集合が保存されている．

四分木が構築される際，ノードに含まれるクエリの数が m 個以下となったノード（葉ノード）は，ノードの id，そのノードに含まれるクエリの id，および根ノードからそのノードまでの経路を表す数字列（ds）をノードリスト（L_N）に保存する．根ノードからそのノードまでの経路を表す数字列は，ノード id を用いて計算し，その結果を保存する．線形四分木では，自身のノード id から 1 を引いたものを 4 で除算した商が親ノードのノード id となり，余りが 0 ならば自身は親ノードの NW というように，余りによって自身は親ノードの NW，NE，SW，および SE のいずれであるかが分かる．この計算を繰り返し行うことにより順に求められた余りを 4 進数の数字列として保存する．例えば，ノード 14 では余りが 1，2 という順に求まるため，12 という数字列を保存する．四分木の構築が完了すると，ノードリスト（L_N）に，クエリが含まれる全ての葉ノード（4a における黒四角）のノード id，そのノードに含まれるクエリの id，および経路を表す数字列を保存する．図 4 において，ノード 15 にはクエリが含まれていないため，L_N を作成する際，ノード 15 は存在していないものとして扱う．



(a) 図 2 に基づく四分木

ノードid	クエリid	数字列
1	0,1	0
2	2	1
13	3	02
14	4	12
16	5	32
4	6,7	3

(b) L_N

図 4: 四分木の構築（$m = 2$）および L_N

3.3 ソーシャルスコアリスト

次に，各 PoI p_i に対応するソーシャルスコアリスト L^i_{ss} の作成について説明する．これは，L_N に保存された各ノードに対して，ノード id およびそのノードに含まれるクエリのソーシャルスコアの中の最大値を保存したものである．具体的には，ノードに含まれている全てのクエリに対して，各 PoI とのソーシャルスコアを計算する．そして，計算したソーシャルスコアの中で最大のものを，そのノードのソーシャルスコア ss_n として L^i_{ss} に保存する．また，ソーシャルスコアは PoI ごとに異なるため PoI ごとに L_{ss} を作成する．しかし，各 L_{ss} に全ての ss_n を保持させる場合，管理コストが膨大になってしまうため，ソーシャルスコアが 0 となるノードのノード id およびソーシャルスコアは保存しない．

3.4 Top-k データの更新アルゴリズム

最後に，ウィンドウのスライドに対して，各クエリの Top-k データを更新するアルゴリズムを説明する．以下では，データの生成および削除に対する処理をそれぞれ説明する．

3.4.1 データの生成

新しくデータが生成されると，L_N，データを生成した PoI の L_{ss}，および四分木を用いて，スコアが j 番目のデータのス

Algorithm 1: NodeCheck($n, ss_n, o, first, last, depth, T_{ss}$)

Input: n：チェック中のノード，ss_n：n のソーシャルスコア，o：データ，$first \cdot last$：チェックに用いる T_{ss} の範囲，$depth$：チェック中のノードの深さ，T_{ss}

```
1: Calculate score_ub
2: if score_min ≦ score_ub then
3:    if first = last //node is a leaf-node then
4:       for ∀q_u ∈ n do
5:          Update(q_u, o, T_ss[first].ds)
6:    else
7:       for ∀n_c ∈ [NW, NE, SW, SE] do
8:          count_{n_c} ← 0
9:       for i = first to last do
10:         if depth-th digit of T_ss[i].ds = 0 then
11:            x ← NW
12:         else if depth-th digit of T_ss[i].ds = 1 then
13:            x ← NE
14:         else if depth-th digit of T_ss[i].ds = 2 then
15:            x ← SW
16:         else
17:            x ← SE
18:         count_x ← count_x + 1
19:         if ss_x < T_ss[i].ss_n then
20:            ss_x ← T_ss[i].ss_n
21:      depth ← depth + 1
22:      for ∀n_c ∈ [NW, NE, SW, SE]
         //NW，NE，SW，SE の順に処理を行う do
23:         if count_{n_c} > 0 then
24:            last ← first + count_{n_c} − 1
25:            NodeCheck(n_c, ss_{n_c}, o, first, last, depth, T_ss)
26:            first ← first + count_{n_c}
27: else
28:    for ∀q_u ∈ n do
29:       if q_u.s_p ≦ score_ub then
30:          q_u.s_p ← score_ub
```

コア以上となる可能性があるクエリ，つまり k-スカイバンドの更新が起こり得るクエリにのみアクセスするためのチェックを行う．

T_{ss} の作成． まず，チェックするノードのソーシャルスコアを得るために用いるソーシャルスコアテーブル T_{ss} を作成する．T_{ss} はデータを生成した PoI の L_{ss} と L_N を組み合わせて作成し，L_N に保存された全てのノードに対して，ノード id，ノードのソーシャルスコア ss_n，および経路を表す数字列を保存する．あるノードの ss_n を得る際，データを生成した PoI の L_{ss} にそのノードが含まれている場合は L_{ss} に保存されている ss_n を，含まれていない場合は 0 を，そのノードの ss_n とする．同時に，T_{ss} に保存される最大のソーシャルスコアを計算し，根ノードの ss_n とする．

ノードのチェック． T_{ss} が作成され，根ノードの ss_n が計算されると，NodeCheck（アルゴリズム 1）を用いて，根ノードから順に，クエリにアクセスすべきかどうかを判断していく．あるノード n におけるクエリにアクセスすべきかどうかの判断は，n に保存された情報，データの位置情報とキーワード集合，および ss_n から計算されたスコアの上界値 $score_{ub}$ と $score_{min}$ との比較により行われる．ここで，$score_{min}$ は，n に含まれるクエリのうち $q_u.score_j$ が最小のものである．以下では，あるノードにアクセスした際の具体的な処理について説明する．

ノードにアクセスすると，まず，$score_{ub}$ を計算する（1 行）．ノード n におけるデータ o のスコアの上界値 $score_{ub}$ は式（5）で計算される．

$$score_{ub} = dist(n, o) + key(n, o) + ss_n \tag{5}$$

$dist(n, o)$ は，データの位置とノード n の間の最小距離を $MAXloc$ で割ったものを 1 から引いた値である．ただし，データの位置がノード n の範囲内に含まれる場合は $dist(n, o) = 1$ である．$key(n, o)$ は，$key(q_u, o)$ を計算する場合と異なり，式（6）によって計算される．

$$key(n, o) = \frac{2|n.key \cap o.key|}{|n.key \cap o.key| + |o.key|} \tag{6}$$

ここで，$n.key$ はノード n が保持しているキーワード集合である．つまり，$key(n, o)$ は，n が保存しているキーワードのうち，データが持つキーワードのみを n が保持していると仮定して，そのキーワードのみで一致度を計算する．

上記により計算される $dist(n, o)$ および $key(n, o)$ は，$q_u \in n.Q$ である全てのクエリ q_u に対して，常に $dist(n, o) \geqq dist(q_u, o)$ および $key(n, o) \geqq key(q_u, o)$ を満たす．また，Top-k データの更新が起こるためには，クエリの k 番目のデータのスコアよりも，新しく発生したデータのスコアが大きくなければならない．ここで，以下の定理が成り立つ．

定理 1. $score_{min} > score_{ub}$ ならば，チェック中のノード n に含まれるクエリにおいて，Top-k データの更新は起こりえない．

証明. n に含まれるクエリ q_u に対して，Top-k データの更新が起きるためには $q_u.score_k \leqq s(q_u, o)$ とならなければいけない．しかし，$score_{min} > score_{ub}$ ならば，$score_{min} > dist(n, o) + key(n, o) + ss_n$ である．また，$q_u.score_k \geqq q_u.score_j \geqq score_{min}$，および $dist(n, o) + key(n, o) + ss_n \geqq dist(q_u, o) + key(q_u, o) + socio(q_u, p) = s(q_u, o)$ であるため，$score_{min} > score_{ub}$ ならば，$q_u.score_k > s(q_u, o)$ である．以上により，定理 1 が証明される． □

$score_{ub}$ が計算されると，$score_{ub}$ とチェック中のノードの $score_{min}$ が比較される（2 行）．$score_{ub}$ の方が小さい場合は，新しく生成されたデータにより，現在チェックしているノードに含まれるクエリの k-スカイバンドの更新が起こり得ないため，チェック中のノードを根とする部分木のチェックを終了し，チェックが終了していないノードのチェックに移る．その際，チェック中のノードに含まれる各クエリに対して，$q_u.s_p$ と $score_{ub}$ を比較し，$score_{ub}$ の方が大きければ $q_u.s_p$ を更新する．ここで，$q_u.s_p$ は枝刈りされたときの $score_{up}$ の最大値を保存したものである．$q_u.s_p$ の詳細については後に説明する．

$score_{ub}$ の方が大きければ，子ノードをチェックしていく．まず，T_{ss} に含まれる葉ノードであり，チェック中のノード n を根とする部分木に含まれる葉ノードが，n の NW，NE，SW，および SE のどのノードの経路上に存在するか計算する．ある葉ノードが n' の経路上に存在するとは，n からその葉ノードにアクセスする際に，n' が中継ノードであることを意味している．この計算は数字列を用いて行い，子ノードの深さ（$depth$）の桁を参照することで計算できる．同時に，n の子ノード（NW，NE，SW，および SE）のソーシャルスコアを計算する（7-18 行）．それらのうち，T_{ss} に含まれる葉ノードである，もしくは子孫に T_{ss} に含まれる葉ノードを 1 つ以上含むノードに対して，NW から順に同様のチェックを行っていく（20-24 行）．チェックしているノードが葉ノードの場合，$score_{ub}$ の方が大きければ，チェック中のノードに含まれるクエリの k-スカイバンドを更新する．そのため，ノードに含まれる全てのクエリに対して，k-スカイバンド，Top-k データ，および四分木に保存されている情報の更新を行うアルゴリズム Update を実行する（3-5 行）.（このアルゴリズムの詳細は，後に説明する.）

例 3.1. 図 4 において，$q_6.score_j = 1.8$ および $q_7.score_j = 2.2$ であるとき，図 2 における p_0 がデータ o を生成したとし，$o.key = \{w_1, w_2, w_5\}$ とする．また，図 4 におけるノード 4 をチェックしたとする．ノード 4 が保存している $n.key = \{w_0, w_1, w_2, w_3, w_4\}$ と $o.key = \{w_1, w_2, w_5\}$ との共通部分は w_1 および w_2 である．このとき，ノード 4 が w_1 および w_2 のみを保持していると仮定して，それらのキーワードとデータ o の持つキーワードの一致度からノード 4 における $key(\cdot, o)$ を計算する．この場合，$key(\cdot, o) = \frac{2\times2}{2+3} = 0.8$ となる．また，PoI p_0 とノード 4 の距離のスコアが $dist(n, o) = 0.6$，ノードのソーシャルスコアが $ss_n = 0.8$ であるとする．このとき，新しく生成されたデータに対するノード 4 におけるスコアの上界値は，$0.6 + 0.8 + 0.8 = 2.2$ となる．計算されたスコアの上界値（2.2）とノード 4 に含まれるクエリの j 番目のデータのスコアの最小値（1.8）を比較すると，スコアの上界値の方が大きい．そのため，ノード 4 に含まれる全てのクエリに対して k-スカイバンドおよび Top-k データの更新を行う．

k-スカイバンドおよび Top-k データの更新． 葉ノードにおいて k-スカイバンドおよび Top-k データの更新が必要であると判断されると，Update（アルゴリズム 2）を用いて，そのノードに含まれる全てのクエリ q_u に対して，k-スカイバンド，Top-k データ，および四分木に保存されている情報の更新を行う．

まず，データのスコアを計算し，$q_u.S$ に含まれるデータ o の $o.dom$ を更新する (1-4 行)．$o.dom$ が k 以上となったデータは $q_u.S$ から削除し，生成されたデータを $q_u.S$ に追加する．次に，新たに生成されたデータのスコアがクエリの k 番目のデータのスコアより大きければ，生成されたデータと k 番目のデータを置き換える（8-9 行）．提案アルゴリズムでは，k-スカイバンドが更新されると，四分木に保存されている情報を更新する必要がある．k-スカイバンドが更新されたクエリを含むノードは，T_{ss} に保存された現在の葉ノードの数字列から計算できる．計算で得られたノードに対して，k-スカイバンドが更新されたクエリの j 番目のデータのスコアの情報を更新する (10-12 行)．

Algorithm 2: Update(q_u, o_{in}, ds)

Input: q_u：クエリ，o_{in}：生成されたデータ，ds：チェック中のノードまでの経路を表す数字列

1: Calculate $s(q_u, o_{in})$
2: **for** $\forall o \in q_u.S$ **do**
3: **if** $s(q_u, o_{in}) \geqq s(q_u, o)$ **then**
4: $o.dom \leftarrow o.dom + 1$
5: **if** $o.dom \geqq q_u.k$ **then**
6: $q_u.S \leftarrow q_u.S - \{o\}$
7: $q_u.S \leftarrow q_u.S + \{o_{in}\}$
8: **if** $s(q_u, o_{in}) \geqq q_u.score_k$ **then**
9: Update $q_u.D$ from $q_u.S$
10: $N \leftarrow$ A set of nodes that are calculated by ds
11: **for** $\forall n \in N$ **do**
12: Update $q_u.score_j$

Algorithm 3: Topk-Reevaluation(q_u, o)

Input: q_u：クエリ，$o : W$ から削除されるデータ

1: $q_u.S \leftarrow q_u.S - \{o\}$
2: **if** $o \in q_u.D$ **then**
3: **if** $|q_u.S| \geqq q_u.k$ **then**
4: Extract $q_u.D$ from $q_u.S$
5: **if** $q_u.s_p \geqq q_u.score_k$ **then**
6: Extract $q_u.D, q_u.S$ from W
7: **else**
8: Extract $q_u.D, q_u.S$ from W

3.4.2 データの削除

新たに生成されたデータに対する処理が終了すると，W から削除されるデータに対する処理を行う（アルゴリズム 3）．まず，削除されるデータを k-スカイバンドに含む全てのクエリ q_u の $q_u.S$ から削除されるデータを取り除く．その中で，削除されるデータを Top-k データに含むクエリに対して，Top-k データの更新を行う．$q_u.S$ に含まれるデータの中で，スコアの高い上位 k 個のデータを Top-k データとする．この際，$|q_u.S| < k$ もしくはスコアが $q_u.s_p$ 以下のデータが Top-k データとなるとき，W 内のデータにおいて $q_u.S$ に含まれないデータが Top-k データとなり得る．ここで $q_u.s_p$ とは，自身を含むノードが枝刈りされたときの $score_{ub}$ の最大値をクエリごとに保存したものである．つまり，スコアが $q_u.s_p$ 以下のデータが Top-k データとなるとき，過去に枝刈りされたデータが Top-k データとなり得る．そのため，W 内の全てのデータに対してスコアを計算し，k-スカイバンドおよび Top-k データの更新を行う．

4. 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．提案アルゴリズム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．

- **ベースライン 1.** データが生成されたとき，全てのクエ

表 1: データセットの詳細

	Yelp	Brightkite
ユーザの数	366,715	50,687
PoI の数	60,785	772,631
全チェックインの数	1,521,160	1,072,965

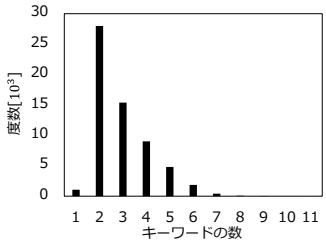


図 5: キーワードの数のヒストグラム

リに対して k-スカイバンドを更新し，Top-k データが削除されたとき，k-スカイバンドを用いて Top-k データを更新する．

- **ベースライン 2.** データが生成されたとき，提案インデックスによりアクセスするクエリの数を削減し，Top-k データが削除されたとき，W 内の全てのデータを再評価する．
- **ベースライン 3.** データが生成されたとき，提案インデックスによりアクセスするクエリの数を削減する．Top-k データが削除されたとき，W 内の全てのデータを再評価する際に L_{ss} を利用する．具体的には，チェックするデータのソーシャルスコアを計算する際に，再計算されるクエリを含む葉ノードがデータを生成した PoI p_i の L^i_{ss} に含まれる場合は保存されたソーシャルスコアを，含まれない場合は 0 とする．そして，計算されたスコアと k 番目のデータのスコアを比較し，Top-k データになり得るかどうかを評価する．Top-k データになり得るデータのみ正確なソーシャルスコアを計算し，Top-k データを更新する．

4.1 セッティング

本実験は，Windows 7，3.47GHz Intel Xeon CPU，および 192GB RAM を搭載した計算機で行い，全てのアルゴリズムは C++で実装した．

データセット. 本実験では，2 つの実データ（Yelp [注2] および Brightkite [注3]）を使用した．表 1 にデータセットの詳細を示す．それぞれのデータセットにおいて，PoI は位置情報を保持しており，ユーザと PoI のソーシャル関係はチェックイン情報を用いた．それぞれのデータセットはキーワードに関する情報を含んでいないため，あるデータ o の $o.key$ として，Yelp のあるレビューにおけるキーワードセットを用いた．このキーワードセットは 1〜11 個のキーワードが含まれている．キーワードの数のヒストグラムを図 5 に表す．

パラメータ. 本実験で用いたパラメータを表 2 に示す．$m = 10$ および $j = 2k$ をデフォルトの値として用いた．ユーザは一人ひとつのクエリを発行するものとし，$q_u.loc$ は各データセットにおいて全ての PoI が存在する領域内の任意の点とした．$q_u.k$ は

(注2) : http://www.yelp.com/dataset_challenge/

(注3) : http://snap.stanford.edu/data/loc-brightkite.html

表 2: パラメータの設定

パラメータ	値（デフォルト）
$\|W\|$ $[\times 10^4]$	50〜300（50）
$q_u.k$ の最大値，k_{max}	5〜30（10）
$\|Q\|$ $[\times 10^4]$	5〜35（5）（Yelp） 1〜5（1）（Brightkite）

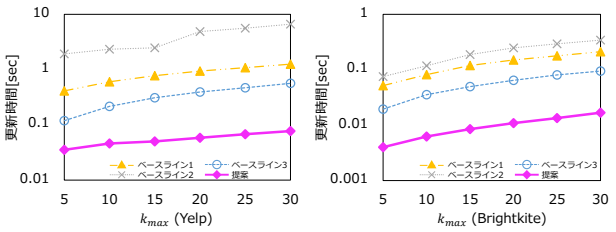
1 から k_{max} の間の一様乱数とし，$q_u.key$ は Yelp のキーワードセットのいずれかを用いた．

4.2 評価結果

本実験では，各アルゴリズムにおける更新時間 :（ウィンドウの各スライドにおいて Top-k データの更新が完了するまでの平均時間 [sec]）の結果を示す．

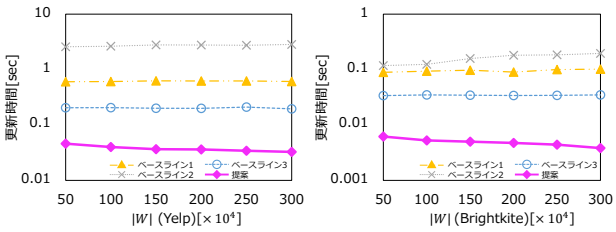
k_{max} の影響. 図 6 に k_{max} を変えたときの結果を示す．まず，各ベースラインアルゴリズムに比べて提案アルゴリズムの方が高速に Top-k データを更新している．提案アルゴリズムはベースラインアルゴリズムと比較して，更新時間が少なくとも 3 分の 1 に抑えられている．k_{max} が大きくなると，よりスコアの低いデータが Top-k データとなるため，各アルゴリズムにおいて更新時間が長くなる．ここで，Yelp と Brightkite の 2 つのデータセットにおいて，Brightkite におけるベースライン 2 と 3 の更新時間の差よりも，Yelp における更新時間の差の方が大きくなる．これは，Yelp はユーザの数に比べて PoI の数が少なく，ユーザが同じ PoI にチェックインしている可能性が大きい．すると，ソーシャルスコアの計算により時間がかかるため，全てのデータに対してソーシャルスコアを計算するベースライン 2 は大きく性能が低下する．

$|W|$ の影響. 図 7 に $|W|$ を変えたときの結果を示す．まず，各ベースラインアルゴリズムに比べて提案アルゴリズムの方が高速に Top-k データを更新している．提案アルゴリズムはベースラインアルゴリズムと比較して，更新時間が少なくとも 4 分の 1 に抑えられている．また，$|W|$ が大きくなると各ベースラインは更新時間が一定もしくは長くなるのに対して，提案アルゴリズムは更新時間が短くなる．ここで，更新時間はデータの生成および削除それぞれに対する処理時間の和である．$|W|$ が大きくなると，よりスコアの大きいデータが Top-k データとなり，提案インデックスを用いると枝刈りが起きやすくなるため，ベースライン 2，3，および提案アルゴリズムではデータの生成に対する処理時間は短くなる．一方，ベースライン 1 では常にすべてのクエリをチェックするためデータの生成に対する処理時間はほぼ一定である．また，$|W|$ が大きくなると，W から削除されるデータが Top-k データに含まれる可能性が低くなり，Top-k データの更新を行う頻度が小さくなる．しかし，ベースライン 2 と 3 において，Top-k データに含まれるデータが削除されると，W 内の全てのデータを再評価するため，$|W|$ が大きくなると再評価するデータが多くなりデータの削除に対する処理時間が長くなる．一方，ベースライン 1 と提案アルゴリズムでは，k-スカイバンドを用いているため，Top-k データに含まれるデータが削除されたときの Top-k データの更新を高速に行



(a) Yelp (b) Brightkite

図 6: k_{max} の影響



(a) Yelp (b) Brightkite

図 7: $|W|$ の影響

うことができる．以上の理由から，提案アルゴリズムでは $|W|$ が大きくなると更新時間が短くなる．

5. 関 連 研 究

本章では，複数の属性を考慮した検索に関する従来研究，また，Pub/Sub モデルにおけるデータのモニタリングに関する従来研究について紹介する．

位置，キーワード，およびソーシャル関係を考慮した検索．位置，キーワード，およびソーシャル関係を考慮した検索についての様々な研究が行われている [1], [7]．文献 [7] では，上記の 3 つの属性に基づく Top-k 検索を効率的に行うインデックスとして，SNIR 木を提案している．SNIR 木の各ノードには，自身を根とする部分木に含まれるデータの位置情報とデータに興味のあるユーザを保存し，クエリが発行されると Top-k データになり得る部分を順に探索していく．しかし，本研究では，ソーシャル関係の定義が異なり，クエリを SNIR 木で管理した場合，各ノードでソーシャルスコアを計算できない．さらに，このデータ構造は Top-k データのモニタリングに対応しておらず，解の更新を行うためには全てのデータを再検索する必要がある．同様に，文献 [1] では，データモニタリングではなくスナップショットクエリを考えており，本研究の問題とは異なる．

Pub/Sub モデルにおけるデータモニタリング．文献 [2], [5] では，Pub/Sub 環境における Top-k データモニタリング手法を提案している．文献 [5] では，位置，およびキーワードに基づいて計算された Top-k クエリの解をスライディングウィンドウ上でモニタリングするためのアルゴリズムを提案している．具体的には，四分木を用いてクエリを管理し，データの生成に対して Top-k データの更新を高速に行う．また，コストベースの k-スカイバンドを用いることにより，データが削除されたとき，Top-k データの再計算を高速に行う．しかし，本研究では，Top-k データを計算する際にソーシャル関係を用いる点，およびデータを生成する PoI ごとにソーシャル関係が変化し，各 PoI ごとに情報を保存する必要がある点から，このアルゴリズムを適用できない．同様に文献 [2] では，Top-k データを計算する際にソーシャル関係ではなく時間を用いている点，およびスライディングウィンドウモデルに適用していない点から，本研究の問題と異なる．

6. 終 わ り に

PoI（Point of Interest）からのパブリッシュ/サブスクライブモデルに基づいたデータの発信，および位置情報サービスやソーシャルネットワークサービスの普及により，位置やキーワード，ソーシャル関係を用いた検索への関心が高まっている．本稿では，パブリッシュ/サブスクライブモデルで生成されたデータの中から，位置，キーワード，およびソーシャル関係に基づき，ユーザにとって有用な上位 k 個のデータをスライディングウィンドウ上でモニタリングする問題に取り組んだ．提案アルゴリズムでは，クエリを四分木により管理し，さらに k-スカイバンドを用いることにより，リアルタイムモニタリングを実現している．評価実験の結果から，提案アルゴリズムの有効性を確認した．

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) 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. Ahuja, N. Armenatzoglou, D. Papadias, and G. J. Fakas. Geo-social keyword search. *SSTD*, pages 431–450, 2015.

[2] L. Chen, G. Cong, X. Cao, and K.-L. Tan. Temporal spatial-keyword top-k publish/subscribe. In *ICDE*, pages 255–266, 2015.

[3] G. Groh and C. Ehmig. Recommendations in taste related domains: collaborative filtering vs. social filtering. In *SIGGROUP*, pages 127–136, 2007.

[4] H. Hu, Y. Liu, G. Li, J. Feng, and K.-L. Tan. A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In *ICDE*, pages 711–722, 2015.

[5] X. Wang, Y. Zhang, W. Zhang, X. Lin, and Z. Huang. Skype: top-k spatial-keyword publish/subscribe over sliding window. *PVLDB*, 9(7):588–599, 2016.

[6] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. Ap-tree: Efficiently support continuous spatial-keyword queries over stream. In *ICDE*, pages 1107–1118, 2015.

[7] D. Wu, Y. Li, B. Choi, and J. Xu. Social-aware top-k spatial keyword search. In *MDM*, pages 235–244, 2014.

[8] C. Zhang, Y. Zhang, W. Zhang, and X. Lin. Inverted linear quadtree: Efficient top k spatial keyword search. In *ICDE*, pages 901–912, 2013.

[9] K. Zheng, H. Su, B. Zheng, S. Shang, J. Xu, J. Liu, and X. Zhou. Interactive top-k spatial keyword queries. In *ICDE*, pages 423–434, 2015.

[10] 西尾俊哉, 天方大地, 原 隆浩. 位置，キーワードおよびソーシャル関係に基づく top-k パブリッシュ/サブスクライブ．第 8 回データ工学と情報マネジメントに関するフォーラム（DEIM フォーラム 2016）, 2016.