DEIM Forum 2019 D5-5

ユーザの移動を考慮した位置・キーワードに基づく Top-k データモニタリング

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，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が配信されており，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，スマートフォンやタブレット端末の普及により，ユーザが移動しながらアプリケーションを利用し，現在地に近いデータを受信するムービングクエリ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の中から，ユーザが移動しながら自身の興味に合う上位 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．データが生成/削除されるたび，すべてのクエリの Top-k データを再評価し，また，ユーザが移動するたび，そのユーザの Top-k データを再評価する方法は非効率的であり，多くのユーザ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，データの生成/削除やユーザの移動より Top-k データが変化する可能性のあるクエリに対してのみ，Top-k データを再評価するアルゴリズムを提案する．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．

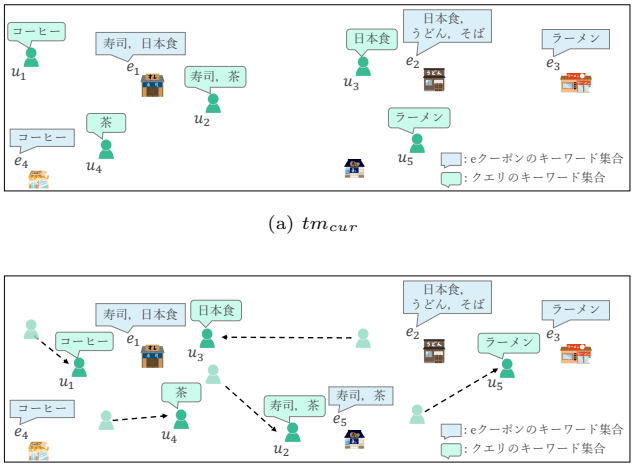
キーワード　Pub/Sub，Top-k クエリ，ムービングクエリ

1. はじめに

近年，SNS や GPS を搭載した端末の普及に伴い，位置情報やキーワードを含むデータが大量に生成されている．これらを検索するアプリケーションの多くは，ユーザが事前に興味のある位置やキーワード（クエリ）を指定し，それら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 [3], [6]．また，データが頻繁に生成/削除される環境では，大量のデータの中から，ユーザの趣向に応じてデータのスコアを計算し，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 Top-k データモニタリングが実用的である [1]．さらに，スマートフォンやタブレット端末の普及により，ユーザが移動しながらアプリケーションを利用することが考えられる．このような場合，指定された位置ではなく，ユーザの現在地に近いデータを検索するムービングクエリが有用である [4], [7]．

これまでに，文献 [1], [5] において，Pub/Sub 環境における Top-k データモニタリングに関する研究が行われている．しかし，これらの研究は，ユーザの指定するクエリの位置は固定であることを想定しており，ユーザの移動を考慮していない．本稿では，Pub/Sub 環境において，ユーザの移動によりクエリの位置が変化するムービングクエリに対する Top-k データをモニタリングする問題に取り組む．

例 1. 図 1 は，Pub/Sub 環境において，ユーザが移動しながら，興味のある e クーポンをモニタリングする例を表している．5 人のユーザがクエリを登録しており，最もクエリに合う e クーポン（Top-1 データ）をモニタリングしている．時刻 tm_{cur} において，ユーザ u_2 の Top-1 データは，u_2 に近くキーワード



(a) tm_{cur}

(b) $tm_{cur}+1$

図 1: Pub/Sub 環境における，ユーザの移動を考慮した Top-k データモニタリングの一例

の類似した e クーポン e_1 であると仮定する．同様に，u_3 の Top-1 データは e_2 であるとする．時刻 $tm_{cur}+1$ において，新たに e_5 が生成され，各ユーザが移動すると，各ユーザの Top-1 データが更新される．具体的には，u_2 の Top-1 データは，e_1 より現在地に近くキーワードの類似した e_5 に変化する．さらに，u_3 は西へ移動したため，u_3 の Top-1 データは，現在地により近い e_1 に変化する．

課題. 各ムービングクエリの Top-k データを効率的にモニタリングするために 2 つの課題が存在する．1 つ目の課題は，頻繁に生成/削除されるデータに対して各クエリの Top-k データをモニタリングすることである．単純な方法では，新たにデータ o が生成されたとき，各クエリ s に対して o のスコアを計算し，

そのスコアが s の現在の k 番目のデータのスコアより良い場合，o が s の Top-k データとなる．さらに，データ o' が削除されたとき，o' を Top-k データに含むクエリ s' の Top-k データを再計算する．これらは，クエリが大量に存在し，データが頻繁に生成/削除される環境では多大な計算コストがかかる．2 つ目の課題は，ムービングクエリに対して Top-k データをモニタリングすることである．あるユーザ u が移動したとき，u が発行したクエリ s の位置が変化することにより，各データのスコアが変化するため，Top-k データが変化する可能性がある．単純な方法では，クエリが移動するたびに全てのデータのスコアを再計算し，Top-k データを更新する．しかし，これには非常に大きな計算コストが必要である．文献 [4] では，ムービングクエリの Top-k データをモニタリングするために Safe Region という技術を提案している．Safe Region は，クエリが移動しても Top-k データが変化しない領域であり，Safe Region を用いることで，クエリが移動したときに Top-k データが変化する場合を把握できる．しかし，データが新たに生成されると，Safe Region が変化する可能性があるため，本問題に単純に適用できない．さらに，クエリが Safe Region の外に出たときや Top-k データが変化したとき，Safe Region を再計算する必要がある．データが大量に存在する環境では，この再計算に多大な計算コストが必要である．

提案アルゴリズムの概要． これらの問題を解決するため，ムービングクエリを Safe Region に基づいて管理する新たな四分木（SQ-tree）を提案する．SQ-tree は，新たにデータが生成されたとき，Top-k データが変化する可能性のあるクエリを効率的に限定する．また，データの削除やユーザの移動により，あるクエリの Top-k データが変化する場合，新たに Top-k データとなるデータを効率的に検索するアルゴリズムを提案する．文献 [10] において筆者らが取り組んだ問題に対して，データの削除を考慮している．さらに，Safe Region の計算コストを削減するため，近似的な Safe Region を高速に計算するアルゴリズムを提案する．

貢献と構成． 以下に，本研究の貢献を示す．(i) 本稿では，Pub/Sub 環境において，移動するユーザ（ムービングクエリ）に対する Top-k データをモニタリングする問題に取り組む（2. 章）．(ii) 効率的に各クエリの Top-k データをモニタリングするアルゴリズムを提案する（3. 章）．(iii) 実データを用いた実験により，提案アルゴリズムの性能を評価する（4. 章）．また，5. 章において関連研究について議論し，6. 章で本稿をまとめる．

2. 予 備 知 識

本章では，本稿で考えるデータモデルおよび問題を詳細に定義し，その後，提案アルゴリズムで利用する既存研究のアルゴリズムを紹介する．

2.1 定　　義

データモデル． あるデータ o は，位置（$o.p$），およびキーワード集合（$o.t$）で構成される（$o = \langle p, t\rangle$）．$o.p$ 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．

クエリモデル． 位置およびキーワードに基づくムービング Top-k クエリを定義する．以降，これを MkSK（**M**oving Top-***k*** **S**patial **K**eyword）クエリと呼ぶ．

定義 1 (MkSK クエリ). ある MkSK クエリ s は，ユーザの現在地（$s.p$），キーワード集合（$s.t$），上位何個のデータを要求するかを指定する変数（$s.k$），および位置とキーワードに対する重みを指定する変数（$s.\alpha$）で構成される（$s = \langle p, t, k, \alpha\rangle$）．$s.t$ は，t_{max} 個以下のキーワードを指定する．o と s が与えられたとき，s に対する o のスコア $score(s, o)$ は以下のように計算される：

$$score(s, o) = \alpha \cdot dist(s.p, o.p) + (1-\alpha)(1-text(s.t, o.t)) \quad (1)$$

ここで，$dist(s.p, o.p)$ はクエリとデータとの空間距離を表すスコア（距離スコア），$text(s.t, o.t)$ はクエリとデータのキーワードの類似度を表すスコア（キーワードスコア）である．また，データ集合 O が与えられたとき，s の Top-k データ A は次の条件を満たす．(i) $|A| = k$, (ii) $\forall o_i \in A, \forall o_j \in O \backslash A, score(s, o_i) \leqq score(s, o_j)$.

今後，特に明記する必要がない場合，$s.k$，および $s.\alpha$ をそれぞれ k，α と表記する．

距離スコアはユークリッド距離を用いて計算され，$dist(s.p, o.p) = \frac{Edist(s.p, o.p)}{Maxdist}$ である．ここで，$Edist(s.p, o.p)$ はクエリとデータとのユークリッド距離，$Maxdist$ はクエリとデータが存在し得る領域 $\mathbb{R}^2$ の最大距離である．キーワードスコアは Jaccard 類似度を用いて計算され，$text(s.t, o.t) = \frac{|s.t \cap o.t|}{|s.t \cup o.t|}$ である．

問題定義． クエリ集合 S およびデータ集合 O が与えられたとき，S に含まれるすべての s の Top-k データをモニタリングする．

2.2 既存研究のアルゴリズム

文献 [4] において，ユーザが移動しても Top-k データが変化しない領域である Safe Region を計算するアルゴリズムが提案されている．まず，Safe Region および Dominant Region を定義する．

定義 2 (Safe Region). O，$s = (p, t, k, \alpha)$，および s の A が与えられたとする．このとき，s の Safe Region R は次の条件を満たす：

$$R = \{p' | \forall o_i \in A, \forall o_j \in O \backslash A, score(s', o_i) \leqq score(s', o_j)\} \quad (2)$$

ここで，$s' = (p', t, k, \alpha)$ である．

定義 3 (Donimant Region). $s = (p, t, k, \alpha)$ および 2 つのデータ o_i, o_j が与えられたとする．このとき，o_i の o_j に対する *Dominant Region* D_{o_i, o_j} は以下の条件を満たす：

$$D_{o_i, o_j} = \{p' | score(s', o_i) \leqq score(s', o_j)\} \quad (3)$$

ここで，$s' = (p', t, k, \alpha)$ である．

これらの定義から，以下の定理が成り立つ [4].

定理 1. O，s，および A が与えられたとき，$R =$

$\cap_{o_i \in A}(\cap_{o_j \in O \setminus A} D_{o_i,o_j})$ である．

Safe Region を効率的に計算するため，文献 [4] では Safe Region の部分領域である Local Safe Region（LSR）を計算する手法を提案している．LSR の計算は 3 ステップで行われる．
ステップ 1 : $o_i \in A$ に対して，以下の条件を満たす楕円領域 $\mathcal{E}_{o_i}$ を計算し，それらの共通部分 $\mathcal{E} = \cap_{o_i \in A} \mathcal{E}_{o_i}$ を計算する．

$$\mathcal{E}_{o_i} = \{p' | dist(s'.p', o_i.p) + dist(s.p, s'.p') \leqq \gamma - r_{o_i}\}, \quad (4)$$

ここで，$r_{o_i} = \frac{1-\alpha}{\alpha}(1 - text(s.t, o_i.t))$，および $\gamma = \max\{dist(s.p, o_i.p) + r_{o_i} | o_i \in A\} + \Delta$ であり，Δ は近似率を決めるパラメータである．また，$\mathcal{E}_{o_i}$ は $s.p$ と $o_i.p$ を焦点とする楕円領域を表す．次に，$s.p$ を中心とし，半径 r_s 内の領域を C_s，および半径 γ 内の領域を C_s^γ とする．また，各データ o_j に対して，$o_j.p$ を中心とし，半径 r_{o_j} 内の領域を C_{o_j} とする．ここで，ある 2 つの領域 C_1，C_2 が与えられたとき，C_1 が C_2 に包含されているとき，C_1 は C_2 内に含まれると定義し，$C_1 \subseteqq C_2$ と表記する．さらに，C_o が C_s^γ に含まれ，C_s に含まれないデータの集合を $O_{C_s^\gamma \setminus C_s}$ とする．つまり，

$$O_{C_s^\gamma \setminus C_s} = \{o | C_o \subseteqq C_s^\gamma, C_o \nsubseteqq C_s\}. \quad (5)$$

である．
ステップ 2 : 各データ $o_j \in O_{C_s^\gamma \setminus C_s}$ に対して，D_{o_i,o_j} を計算し，それらの共通部分 $D = \cap_{o_i}(\cap_{o_j \in C_s^\gamma \setminus C_s} D_{o_i,o_j})$ を計算する．
ステップ 3 : E と D の共通部分を計算する．

時間計算量．ステップ 1 は，k 個の楕円領域およびそれらの共通部分を計算するため，時間計算量は $\mathcal{O}(k^2)$ である．次に，頂点の数が v_1 および v_2 個である 2 つの多角形が与えられたとき，それらの共通部分の計算の時間計算量は $\mathcal{O}((v+l)\log_2(v+l))$ である [8]．ここで，$v = v_1 + v_2$ および l は多角形の辺同士の交点の数である．よって，ステップ 2 の時間計算量は $\mathcal{O}(ck|O_{C_s^\gamma \setminus C_s}|)$ である．ここで，$c = (v+l)\log_2(v+l)$ である．ステップ 3 の時間計算量は $\mathcal{O}(c)$ である．したがって，Local Safe Region の計算の時間計算量は $\mathcal{O}(ck|O_{C_s^\gamma \setminus C_s}|)$ である．

3. 提案アルゴリズム

各クエリの Top-k データは，ユーザの移動およびデータの生成/削除により変化する可能性がある．ここから，ユーザの移動，データの生成，およびデータの削除に対する処理の詳細を紹介する．

3.1 ユーザの移動に対する処理

提案アルゴリズムでは，Safe Region を用いて，各クエリ s の Top-k データをモニタリングする．2.2 節で紹介した LSR の計算には，$\forall o_i \in s.A$ および $\forall o_j \in O_{C_s^\gamma \setminus C_s}$ に対して D_{o_i,o_j} を計算する必要がある．多くのデータが存在している環境では，$O_{C_s^\gamma \setminus C_s}$ に多くのデータが含まれるため，LSR の計算コストは非常に大きくなる．この問題を解決するため，データの分布によらず近似的な Safe Region（ASR）を計算する手法を提案する．まず，s に対する $k+1$ 番目のデータを o_{k+1} とする．そして，$o_i \in A$ に対して $\mathcal{E}_{o_i}$ を計算するとき，$\gamma = dist(s.p, o_{k+1}.p) + r_{o_{k+1}}$ とする．このとき，以下の定理が成り立つ．

定理 2. $O_{C_s^\gamma \setminus C_s} = \varnothing$

本稿では，スペースの都合上，すべての証明を省略する．

定理 2 より，ASR は，Safe Region を計算する際に，Dominant Region を計算する必要がない．つまり，楕円領域およびそれらの共通部分を計算するだけで Safe Region を計算できるため，Safe Region の計算コストを削減できる．今後，Safe Region は ASR を指すとする．つまり，$R = \mathcal{E}$ である．

時間計算量．ASR の計算は，k 個の楕円領域およびそれらの共通部分を計算するのみであるため，時間計算量は $\mathcal{O}(k^2) \ll \mathcal{O}(ck|O_{C_s^\gamma \setminus C_s}|)$ である．

3.2 データの生成に対する処理

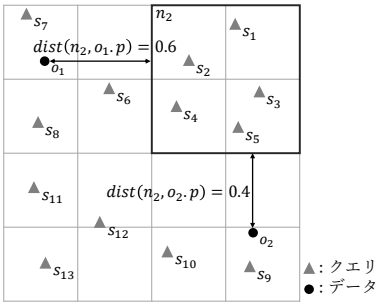
新たなデータ o が生成されたとき，各クエリの Top-k データを高速に更新するため，o により Top-k データが変化する可能性のあるクエリ集合 S_g を限定する必要がある．これを実現する方法として，R-tree や Quad-tree のような空間インデックスでクエリを管理することが考えられる．しかし，ムービングクエリはクエリの位置が常に変化するため，これらのインデックスで単純に管理できない．この問題を解決するため，ムービングクエリを Safe Region に基づいて管理する新たな四分木（SQ-tree）を提案する．そして，新たにデータが生成されたとき，SQ-tree を用いて Top-k データが変化する可能性のあるクエリを効率的に限定する．

SQ-tree の構造．あるクエリ s の R の重心を g とする．SQ-tree の各ノードは，g がノードの示す領域に含まれるクエリを管理する．また，SQ-tree は，既存の空間インデックスとは異なり，葉ノードだけでなく，中継ノードでもクエリを管理し，あるノード n で管理するクエリの集合を S_n，n を根とする部分木で管理されるクエリ集合を S_n^r とする．n は，S_n^r に含まれるクエリのキーワード集合 $T_n^r = \cup_{s \in S_n^r} s.t$，および新たに発生したデータ o により S_n^r に含まれるすべてのクエリの Top-k データが変化することのない距離スコアの閾値 $\Lambda_n^r[i]$ $(0 \leqq i \leqq t_{max})$ を保存する．さらに，$S_n \neq \varnothing$ であるとき，n は，S_n に含まれるクエリのキーワード集合の転置ファイル I_n，および S_n に含まれるクエリに対する距離スコアの閾値 $\Lambda_n[i]$ を保存する．$\Lambda_n^r[i]$ および $\Lambda_n[i]$ の詳細は後述する．

例 2. 図 2 に示すようにクエリが存在し，三角形が Safe Region の重心を示すと仮定すると，図 3 の左図に示す SQ-tree が構築される．例えば，$S_{n_2}^r = \{s_1, s_2, s_3, s_4, s_5\}$ であるため，n_2 はそれらのキーワード集合および $\Lambda_n^r[\cdot]$ を保存する．さらに，$S_n = \{s_1, s_3\}$ であるため，n_2 はそれらのキーワード集合の転置ファイルおよび $\Lambda_n[\cdot]$ を保存する．

まず，SQ-tree を用いて Top-k データが変化しないクエリを枝刈りするフィルタリング技術を紹介し，その後，SQ-tree の構築方法およびメンテナンス方法について紹介する．

フィルタリング技術．あるクエリ s の k 番目のデータのスコアを $score_k(s)$ とする．このとき，新たに発生したデータ o により，s の Top-k データが更新されるためには，$score_k(s) > score(s, o)$



クエリ	キーワード
s_1	w_1, w_2
s_2	w_3
s_3	w_2, w_4
s_4	w_3, w_4, w_5, w_6
s_5	w_2, w_5
...	...

データ	キーワード
o_1	w_2, w_7, w_8
o_2	w_1, w_2, w_3, w_5

図 2: クエリおよびデータの位置とキーワード

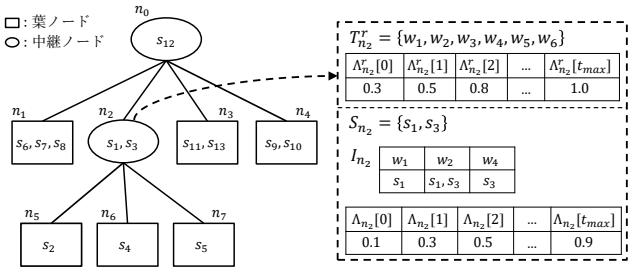


図 3: SQ-tree の例

が必要である．ここで，位置，およびキーワードを考慮した検索において，キーワードを優先したフィルタリングは，高速化に大きく貢献することが知られている [2]．そこで，提案アルゴリズムでは，クエリとデータの間で一致するキーワードの数に基づいたフィルタリング技術を提案する．

まず，$s.p$ が既知であると仮定し，$s.t$ と $o.t$ の間で一致するキーワードの個数が i であるとき，$text(s.t, o.t)$ の上界値 $text_{ub}(s.t, i)$ は以下で与えられる．

補助定理 1. $text_{ub}(s.t, i) = \frac{i}{|s.t|}$.

補助定理 1 および式（1）より，Top-k データが更新されるための距離スコアの閾値 $\lambda_d(s, i)$ が以下のように計算できる：

$$\lambda_d(s, i) = \frac{score_k(s)}{\alpha} - \frac{1-\alpha}{\alpha}(1 - text_{ub}(s.t, i)). \quad (6)$$

式（6）より，以下の定理が成り立つ．

定理 3. $\lambda_d(s, i) < dist(s.p, o.p)$ ならば，o は s の Top-k データになり得ない．

定理 3 より，$\lambda_d(s, i) < dist(s.p, o.p)$ ならば，正確性を失うことなく s を枝刈りできる．

本研究では，ユーザは常に移動しているため，$s.p$ は常に変化する．このとき，$s.p$ により $score_k(s)$ が変化し，$\lambda_d(s, i)$ が変化するため，正確に枝刈りできない．この問題を解決するため，$s.p$ が R 内の任意の位置であるとき，正確に枝刈りが可能なフィルタリング技術を提案する．まず，$s.p$ が R 内の任意の位置であるとき，$score_k(s)$ の上界値 $score_{ub}(s)$ は以下で与えられる．

補助定理 2. $score(s, o_i) \leqq \frac{\gamma \cdot \alpha + score_k(s)}{2} = score_{ub}(s)$

式（6）および補助定理 2 より，$s.p$ が R 内の任意の位置であるとき，Top-k データが更新されるための距離スコアの閾値 $\lambda(s, i)$ は以下のように計算できる：

$$\lambda(s, i) = \frac{score_{ub}(s)}{\alpha} - \frac{1-\alpha}{\alpha}(1 - text_{ub}(s.t, i)). \quad (7)$$

式（7）より，以下の定理が成り立つ．

定理 4. $\lambda(s, i) < dist(s.R, o.p)$ ならば，o は s の Top-k データになり得ない．ここで，$dist(s.R, o.p)$ は，$o.p$ と $s.R$ 内の任意の点との距離スコアの最小値である．

さらに，定理 4 から，以下の系が成り立つ

系 1. $\lambda(s, i) < dist(s.R, o.p)$ ならば，o により $s.R$ は変化しない．

定理 4 および系 1 より，$\lambda(s, i) < dist(s.R, o.p)$ ならば，正確性を失うことなく s を枝刈りでき，$s.R$ の更新も考えなくてよい．

ここから，SQ-tree の各ノードで管理されているクエリを枝刈りする方法について紹介する．この枝刈りは，ノードフィルタリングおよびクエリフィルタリングの 2 段階の枝刈りを行う．新たにデータ o が生成され，ノード n をチェックしたと仮定し，それぞれの枝刈りについて説明する．まず，$o.p$ と n との距離スコアの最小値を $dist(n, o.p)$ とし，$o.p$ が n の領域内に含まれる（$o.p \in n$）ならば，$dist(n, o.p) = 0$ とする．また，R が n に包含される場合，R は n に含まれる（$R \subseteqq n$），包含されない場合，R は n に含まれない（$R \nsubseteqq n$）と定義する．$R \nsubseteqq n$ であるとき，$dist(n, o.p) \leqq dist(R, o.p)$ が常に成り立つとは限らない．この問題を解決するため，R が n からどれだけはみ出ているかを表す追加の距離スコア $dist_{add}(R, n)$ を考える．$dist_{add}(R, n)$ は以下の式で与えられる．

$$dist_{add}(R, n) = \max(0, dist_{max}(g, \hat{R}) - dist_{min}(g, \hat{n})). \quad (8)$$

ここで，$dist_{max}(g, \hat{R})$ は g と R の境界 $\hat{R}$ との距離スコアの最大値，$dist_{min}(g, \hat{n})$ は g と n の境界 $\hat{n}$ との距離スコアの最小値である．このとき，以下の不等式が成り立つ．

$$dist(n, o.p) \leqq dist(R, o.p) + dist_{add}(R, n) \quad (9)$$

式（9）より，$\Lambda^r_n[i]$ および $\Lambda_n[i]$ をそれぞれ以下で定義する．

$$\Lambda^r_n[i] = \max\{\lambda(s, i) + dist_{add}(s.R, n) | s \in S^r_n\},$$
$$\Lambda_n[i] = \max\{\lambda(s, i) + dist_{add}(s.R, n) | s \in S_n\}.$$

ここで，$|s.t| < i$ ならば，$\lambda(s, i) = \lambda(s, |s.t|)$ である．

クエリフィルタリング. ノード n で管理されているクエリを枝刈りする方法を紹介する．まず，$\Lambda_n[i] \geqq dist(n, o.p)$ を満たす最小の i を i_{min} とすると，以下の定理が成り立つ．

定理 5. o，i_{min}，および $s \in S_n$ が与えられたとき，$|s.t \cap o.t| < i_{min}$ ならば，o は s の Top-k データになり得ない．

定理 5 より，正確性を失うことなく，$|s.t \cap o.t| < i_{min}$ を満たすクエリを枝刈りでき，それらの Top-k データおよび Safe Region を更新する必要がない．

アルゴリズム 1 は，クエリフィルタリングを行うアルゴリズムを示している．まず，i_{min} を計算する（2 行）．ここで，T^r_n と $o.t$ の間で一致するキーワードの数を i_c とし，$i_c \geqq t_{max}$ ならば，$i_c = t_{max}$ とする．$i_c < i_{min}$ ならば，$s \in S_n$ に対して，

Algorithm 1: QueryFiltering(o,$dist(n, o.p)$,i_c)

Input: o, $dist(n, o.p)$, and i_c

1 $S_g \leftarrow \varnothing$ //S_g is a set of queries whose top-k results may change.
2 $i_{min} \leftarrow \min\{i|\Lambda_n[i] \geqq dist(n, o.p)\}$
3 **if** $i_{min} \leqq i_c$ **then**
4 **for** $\forall s \in S_n$ *where* $|s.t \cap o.t| \geqq i_{min}$ **do**
5 $S_g \leftarrow S_g \cup \{s\}$
6 return S_g

$|s.t \cap o.t| < i_{min}$ が成り立つため，$i_{min} \leqq i_c$ である場合のみ，$|s.t \cap o.t| \geqq i_{min}$ を満たす s を S_g に加える（3–5 行）．

例 3. 図 2 および図 3 において，o_2 が生成され，n_2 に対して，QueryFiltering$(o_2, dist(n_2, o_2.p), i_c)$ を実行したと仮定する．ここで，$i_c = |T^r_{n_2} \cap o_2.t| = 4$ である．$dist(n_2, o_2.p) = 0.4$ であるため，$i_{min} = 2$ である．また，$|s_1.t \cap o_2.t| = 2$ および $|s_3.t \cap o_2.t| = 1$ である．よって，s_1 のみ S_g に加える．

ノードフィルタリング．ここから，n を根とする部分木を枝刈りする方法を紹介する．T^r_n は，S^r_n に含まれるクエリのキーワード集合であるため，$i_c = \min\{|T^r_n \cap o.t|, t_{max}\}$ が与えられたとき，以下の定理が成り立つ．

定理 6. $\Lambda^r_n[i_c] < dist(n, o.p)$ ならば，o は S^r_n に含まれるすべてのクエリの Top-k データになり得ない．

定理 6 より，$\Lambda^r_n[i_c] < dist(n, o.p)$ ならば，正確性を失うことなく，n を枝刈りでき，S^r_n に含まれるすべてのクエリの Top-k データおよび Safe Region を更新する必要がない．

アルゴリズム 2 は，ノードフィルタリングを行うアルゴリズムを示している．まず，i_c を計算する（2 行）．n が枝刈りされずかつ葉ノードでないとき，n の子ノード n' に対して Nodefiltering(n', o) を再帰的に実行する．さらに，$S_n \neq \varnothing$ ならば，SubscriptionFiltering$(o, dist(n, o.p), i_c)$ を実行する（4–8 行）．

例 4. 図 2 および図 3 において，o_1 が生成され，n_2 に対して，Nodefiltering(n_2, o_1) を実行したと仮定する．$T^r_{n_2}$ と $o_1.t$ の間で一致するキーワードは w_2 のみであるため，$i_c = 1$ である．$dist(n_2, o_1.p) = 0.6$ であるため，$\Lambda^r_{n_2}[i_c] < dist(n_2, o_1.p)$ である．よって，n_2 は枝刈りされる．つまり，$S^r_{n_2}$ に含まれるすべてのクエリの Top-k データおよび Safe Region を更新しない．

SQ-tree の根ノードからノードフィルタリングを実行することにより，S_g を効率的に得ることができる．

解の更新．フィルタリングによって得られた S_g に含まれる各クエリ s に対して Top-k データおよび Safe Region を更新する．まず，s を発行したユーザに現在地を要求した後，各 Top-k データに対するスコアを更新し，さらに，$score(s, o)$ を計算する．$score(s, o) < score_k(s)$ ならば，s の Top-k データを更新し，更新前に k 番目であったデータを o_{k+1} とする．一方，$score(s, o) \geqq score_k(s)$ ならば，s の Top-k データは変化しない．しかし，s の Safe Region を再計算する必要があり，さらに，必ずしも o が o_{k+1} であるとは限らないため，o_{k+1} を検索する必要がある．o_{k+1} を検索する方法については，3.3 節において紹介する．o_{k+1} を検索した後，s の新たな Safe Region を再計算する．

Algorithm 2: NodeFiltering(o, n)

Input: n and o

1 $S_g \leftarrow \varnothing$
2 $i_c \leftarrow \min(|T^r_n \cap o.t|, t_{max})$
3 **if** $\Lambda^r_n[i_c] \geqq dist(n, o.p)$ **then**
4 **if** n *is not a leaf node* **then**
5 **for** $\forall n' \in N$ //N *is* n*'s children* **do**
6 $S_g \leftarrow S_g \cup$ NodeFiltering(n', o)
7 **if** $S_n \neq \varnothing$ **then**
8 $S_g \leftarrow S_g \cup$ SubscriptionFiltering$(o, dist(n, o.p), i_c)$
9 return S_g

SQ-tree の構築．スコアは，位置，およびキーワードに基づいて計算されるため，両者を考慮して，クエリを分散するのが望ましい．しかし，R-tree や Quad-tree といった空間インデックスは，位置に基づいて領域を分割していく木構造であるため，キーワードの特徴に基づいてクエリを分散できない．この問題を解決するため，SQ-tree は，すべてのクエリを葉ノードで管理するのではなく，各クエリのキーワードに関する特徴を考慮し，中継ノードでもクエリを管理する．ある中継ノード n でクエリを管理した場合，n の子ノード n' に対して，$S^r_{n'}$ に含まれるクエリの数が減少し，n' の距離スコアの閾値 $\Lambda^r_n[\cdot]$ がタイトになる．その結果，ノードフィルタリングにより n' を枝刈りできる可能性が大きくなる．

ここから，あるクエリ s をノード n で管理すべきか，子ノードに分配すべきかを決定するアルゴリズムについて紹介する．まず，あるデータ o が生成されたとき，n に対して計算される $dist(n, o.p)$ の期待値を $E[dist(n, o.p)]$(注1) とする．このとき，式（1）および補助定理 2 から，s の Top-k データが更新されるために必要なキーワードスコアの期待値 $E[text(s.t, o.t)]$ は以下のように計算される：

$$E[text(s.t, o.t)] = 1 - \frac{score_{ub}(s) - \alpha \cdot E[dist(n, o.p)]}{1 - \alpha}. \tag{10}$$

また，s の Top-k データが更新されるために必要な $|s.t \cap o.t|$ の期待値 $E[|s.t \cap o.t|]$ は以下のように計算される．

$$E[|s.t \cap o.t|] = \lfloor E[text(s.t, o.t)] \cdot |s.t| \rfloor. \tag{11}$$

ここで，データのキーワードの分布が時間により変化しないと仮定すると，T^r_n と $o.t$ の間で一致するキーワードの数の期待値 $E[|T^r_n \cap o.t|]$ は，以下の式で与えられる．

$$E[|T^r_n \cap o.t|] = \min(t_{max}, \sum_{w_i \in T^r_n} \frac{|O_{w_i}|}{|O|}) \tag{12}$$

ここで，O_{w_i} は w_i を含むデータの集合である．新たにデータ

(注1)：$E[dist(n, o.p)] = \frac{1}{S} \iint_{\mathbb{R}^2} dist(n, o.p) dx dy$ （S：領域 $\mathbb{R}^2$ の面積）

o が生成されたとき，$E[|T_n^r \cap o.t|] < E[|s.t \cap o.t|]$ ならば，s が枝刈りされる可能性が大きい．よって，s がこの条件を満たす場合，子ノードへ分配せず n で管理する．

SQ-tree のメンテナンス．クエリ s の Top-k データおよび Safe Region が更新されたとき，s を管理するノードの情報が変化する可能性があるため，SQ-tree を更新する必要がある．更新後の Safe Region の重心を g_{new} とする．s を管理する n に対して，$g_{new} \in n$ ならば，単純にノードに保存された情報を更新する．一方，$g_{new} \notin n$ ならば，s を一旦 SQ-tree から取り除き，更新後の Safe Region に基づいて SQ-tree に再度挿入する．

3.3 データの削除に対する処理

あるデータ o' が削除されたとき，o' を Top-k データに含むクエリ集合 S_e に対して，Top-k データおよび Safe Region を更新する必要がある．$s \in S_e$ の Top-k データおよび Safe Region を更新するため，s に対するスコアが k 番目および $k+1$ 番目であるデータ o_k および o_{k+1} を検索する必要がある．ここで，あるデータ o が s の Top-k データであるとき，o は次の条件のいずれかを満たす : (1) $o \in O_{kNN}$，(2) $o \in O_{s.t}$．ここで，O_{kNN} は，$s.p$ から近い k 個のデータ集合，$O_{s.t}$ は，$s.t \cap o.t \neq \varnothing$ を満たすデータ集合である．

これらの条件を満たすデータを効率的に検索するため，提案アルゴリズムにおけるデータのインデックスでは，データの位置情報を管理する Kd-tree とデータのキーワード集合を管理する転置ファイルを用いる．Kd-tree を用いて O_{kNN} を効率的に検索でき，転置ファイルを用いて $O_{s.t}$ を効率的に検索できる．しかし，あるキーワード $w_i \in s.t$ が多くのデータに含まれる場合，w_i のポスティングリスト $I_O[w_i]$ に含まれるすべてのデータをチェックするのは多大な時間がかかる．この問題を解決するため，文献 [9] と同様に，データのキーワードごとに位置情報を管理する四分木を作成する．しかし，すべてのキーワードに対して四分木を作成すると膨大なメモリコストが必要である．そのため，すべてのキーワードのうち，出現頻度が上位 x% 以上のキーワードを頻出キーワードとし，頻出キーワード w_i に対して四分木 Q_{w_i} を作成する．図 4 は，データのインデックスの一例を示しており，w_1 および w_2 が頻出キーワードであるため，Q_{w_1} および Q_{w_2} を作成する．

ここから，o_{k+1} を検索する方法を紹介する．(o_k を検索する必要がある場合，同様の方法で検索すればよい．) まず，Top-k データに含まれない各データ $o \in O \backslash s.A$ に対して，$|s.t \cap o.t| = i$ ならば，$score(s,o)$ の下界値 $score_{lb}(s,i,o_{nn})$ は以下の式で計算される．

$$score_{lb}(s,i,o_{nn}) = \alpha \cdot dist(s.p, o_{nn}.p) + (1-\alpha)(1 - \frac{i}{|s.t|}) \quad (13)$$

ここで，o_{nn} は，$O \backslash s.A$ に含まれるデータの中で $s.p$ に最も近いデータである．また，これまでにチェックされたデータの中で，最もスコアの良いデータを暫定の o_{k+1} とすると，以下の定理が成り立つ．

定理 7. $score_{lb}(s,i,o_{nn}) \geqq score(s,o_{k+1})$ ならば，$|s.t \cap o.t| \leqq i$ を満たすデータ $o \in O_{s.t} \backslash s.A$ は o_{k+1} になり得ない．

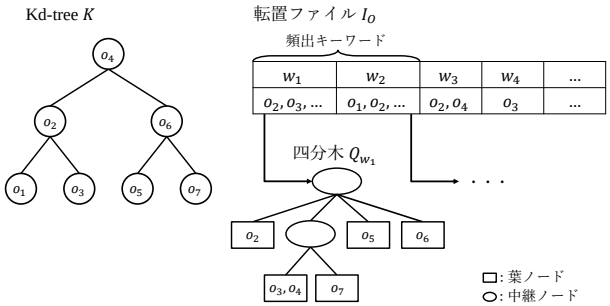


図 4: データのインデックスの例

定理 7 より，$score_{lb}(s,i,o_{nn}) \geqq score(s,o_{best})$ ならば，正確性を失うことなく，$|s.t \cap o.t| < i$ を満たすデータを枝刈りできる．

アルゴリズム 3 は，o_{k+1} を検索するアルゴリズムを示している．まず，Kd-tree を用いて，$O \backslash s.A$ の中で $s.p$ に最も近いデータを返す FindNearestObject$(s.p, s.A)$ を実行し，o_{nn} を検索する．そして，暫定的に o_{nn} を o_{k+1} とする（1–2 行）．$|s.t \cap o.t| = i$ の大きいデータが o_{k+1} となる可能性が大きいため，$O_{s.t}$ に含まれるデータを i の降順でチェックする．$i > 1$ のとき，$|s.t \cap o.t| = i$ を満たすデータ $o \in O_{s.t}$ に対して，CheckObject(s,o,o_{k+1}) を実行する（7–9 行）．ここで，CheckObject(s,o,o_{k+1}) は，$o \notin s.A$ かつ $score(s,o) < score(s,o_{k+1})$ ならば，o を o_{k+1} とする．一方，$i = 1$ のとき，各クエリキーワード $w_i \in s.t$ に対して，w_i が頻出キーワードである場合，CheckQtree(Q_{w_i}, s, o_{k+1}) を実行し，頻出キーワードでない場合，$I_O[w_i]$ に含まれるすべてのデータに対して CheckObject(s,o,o_{k+1}) を実行する（10–16 行）．CheckQtree(Q_{w_i}, s, o_{k+1}) は，四分木 Q_{w_i} に含まれるデータを $s.p$ に近い順にチェックし，$o \notin s.A$ かつ $score(s,o) < score(s,o_{k+1})$ ならば，o を o_{k+1} とする．チェック済みでないデータのスコアが $score(s,o_{k+1})$ を下回ることがなくなった場合，検索を終了する．

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．本実験は，Windows 10 Pro，3.20GHz Intel Core i7，および 64GB RAM を搭載した計算機で行い，すべてのアルゴリズムは C++で実装した．

4.1 セッティング

本実験では，提案アルゴリズムのいくつかの機能を用いた手法 : A-KIF，A-SQ-KI，および A-SQ-IQ との比較を行った．ここで，A および SQ は，提案アルゴリズムにおける近似的な（**A**pproximate）Safe Region および **SQ**-tree を示し，KIF は，提案アルゴリズムにおけるデータのインデックス（**K**d-tree + 転置（**I**nverted）ファイル+頻出（**F**requent）キーワードに対する四分木）を示す．さらに，KI は，Kd-tree および転置ファイルのみを用いたデータのインデックス，IQ は，IR-tree のように四分木と転置ファイルを組み合わせた木構造である **IQ**-tree を示す．つまり，A-KIF は，近似的な Safe Region および KIF のみを用いた手法であり，新たにデータが生成されたとき，すべてのクエリの Top-k データの更新をチェックする．A-SQ-KI

Algorithm 3: RetrieveObject(s)

Input: s, O

```
o_nn ← FindNearestObject(s.p, s.A)
o_{k+1} ← o_nn
for ∀o ∈ O_{s.t} do
    Compute |s.t ∩ o.t| by using Inverted file I_O
i ← |s.t|
while i > 0 ∧ score_lb(s, i, o_nn) < score(s, o_{k+1}) do
    if i > 1 then
        for ∀o ∈ O_{s.t} such that |s.t ∩ o.t| = i do
            CheckObject(s, o, o_{k+1})
    else
        for ∀w_i ∈ s.t do
            if w_i is a frequent keyword then
                CheckQtree(Q_{w_i}, s, o_{k+1})
            else
                for ∀o ∈ I_O[w_i] do
                    CheckObject(s, o, o_{k+1})
    i ← i − 1
return o_{k+1}
```

は，頻出キーワードに対して四分木を作成しないため，アルゴリズム 3 において CheckQtree を実行せず，常に CheckObject を実行する．A-SQ-IQ は，IQ-tree を用いて o_{k+1} を検索する．

データセット. 本実験では，生成されるデータとして Twitter [1] のツイートおよび Yelp(注2) のレビューを用いた．表 1 に，データセットの詳細を示す．各クエリは，1 つのツイートまたはレビュー中に現れる単語の中から，t_{max} 個以下の単語をランダムに選び，クエリのキーワード集合とする．また，ユーザのトラジェクトリとして，ATC(注3) および OSM(注4) を用いた．各トラジェクトリは，1 秒を 1 タイムスタンプ（tm）として，100 個の連続する点を含むトラジェクトリを抽出し，ユーザのトラジェクトリとした．

パラメータ. 表 2 に，本実験で用いたパラメータを示す．太字で表されている値はデフォルトの値である．また，$t_{max} = 5$ および $x = 1$ とし，Twitter（Yelp）を用いたとき $m = 1000(100)$ とした．さらに，k は 1 から k_{max} の間の一様乱数とし，α は 0 から 1 の間の一様乱数とした．実験は，データが 1,000,000 個発生した時点から開始する．

4.2 評価結果

LSR と ASR の計算時間の比較. 図 5 に LSR および ASR の平均計算時間の結果を示す．ASR は，LSR と比較して，非常に高速に Safe Region を計算できることが分かる．これは，ASR は，Dominant Region およびそれらの共通部分を計算することなく，Safe Region を計算できるためである．

ここから，各アルゴリズムにおける平均更新時間（tm あたりの各クエリの Top-k データの更新が完了するまでの平均時間 [msec]）の結果を示す．

(注2) : http://www.yelp.com/dataset_challenge
(注3) : http://www.irc.atr.jp/crest2010_HRI/ATC_dataset/
(注4) : https://planet.openstreetmap.org/gps/

表 1: データセットの詳細

	Twitter	Yelp
キーワードの数	292,193	836,680
1 つのデータに含まれるキーワードの数の平均	4.6	29.6

表 2: パラメータの設定

パラメータの設定	値
$s.k$ の最大値，k_{max}	5, **10**, 15, 20
クエリの数，$\|S\|$ [$\times 10^6$]	**0.25**, 0.5, 0.75, 1
データの生成頻度，$f_g(/tm)$	1, **5**, 10, 20
データの削除頻度，$f_e(/tm)$	1, **5**, 10, 20

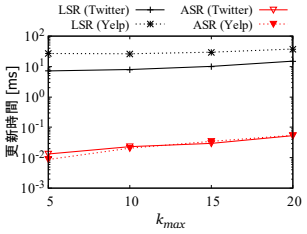
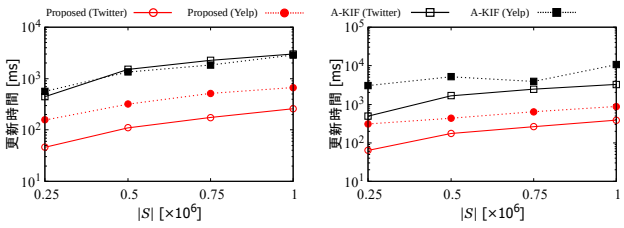


図 5: LSR と ASR の計算時間の比較

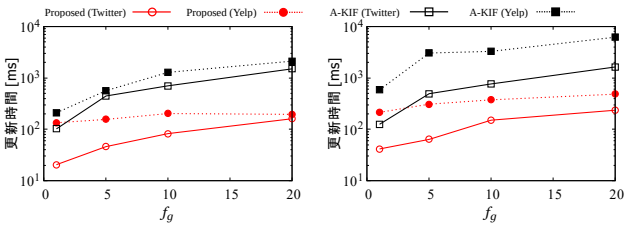


(a) ATC　　(b) OSM

図 6: $|S|$ の影響

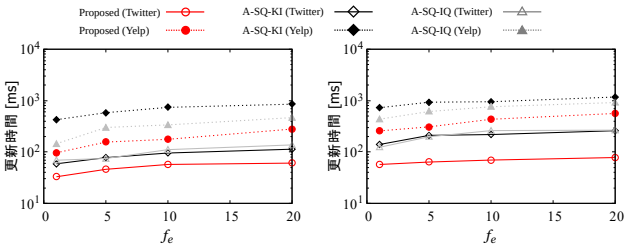
$|S|$ の影響. 図 6 に $|S|$ を変えたときの結果を示す．提案アルゴリズムは A-KIF と比較して，高速に Top-k データを更新できることが分かる．また，$|S|$ の増加に伴い，両手法の更新時間の差が大きくなることが分かる．新たにデータが生成されたとき，A-KIF は常にすべてのクエリの Top-k データの更新をチェックするのに対し，提案アルゴリズムは，Top-k データが変化する可能性のあるクエリを限定し，それらのみ Top-k データの更新をチェックする．その結果，$|S|$ の増加に伴い Top-k データの更新をチェックするクエリの数の差が大きくなるため，更新時間の差が大きくなる．

f_g の影響. 図 7 にデータの生成頻度 f_g を変えたときの結果を示す．f_g の増加に伴い，両手法において更新時間が大きくなることが分かる．また，提案アルゴリズムは A-KIF と比較して，高速に Top-k データを更新できることが分かる．両手法は新たに発生するデータに対して 1 つずつ処理を行うため，データの生成に対する更新時間は，線形に増加する．また，f_g が増加すると，ユーザが自身の Safe Region の外に出る可能性が大きくなり，Top-k データおよび Safe Region の再計算の頻度が増加する．以上の理由から，f_g が増加すると，更新時間が増加する．



(a) ATC (b) OSM

図 7: f_g の影響



(a) ATC (b) OSM

図 8: f_e の影響

f_e の影響．図 8 にデータの削除頻度 f_e を変えたときの結果を示す．提案アルゴリズムはその他の手法と比較して，高速に Top-k データを更新できることが分かる．また，f_e の増加に伴い，各手法において更新時間が大きくなることが分かる．f_e が増加すると，よりスコアの大きいデータが Top-k データとなるため，新たに生成されたデータが Top-k データとなる可能性が大きくなり，また，データの削除による Top-k データおよび Safe Region の更新の頻度が増加する．その結果，更新時間が大きくなる．

5. 関 連 研 究

本章では，Pub/Sub 環境における Top-k データモニタリングに関する従来研究，およびムービングクエリに関する従来研究について説明する．

Pub/Sub 環境における Top-k データモニタリング．文献 [1], [5] において，Pub/Sub 環境における Top-k データモニタリング手法が提案されている．これらの研究では，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．しかし，これらの研究では，クエリの位置は固定であることを想定しており，ユーザの移動に対応できない．

ムービングクエリ．これまでに多くの研究がムービングクエリの解をモニタリングする問題に取り組んでいる．文献 [4], [7] では，ムービング Top-k クエリの解を効率的にモニタリングするアルゴリズムが提案されている．具体的には，クエリが移動しても，Top-k データが変化することのない領域である Safe Region を計算することで，クエリが Safe Region の外に出た場合のみ Top-k データの更新を行う．しかし，これらの研究はストリームデータを対象としておらず，新たにデータが生成された場合，Top-k データおよび Safe Region を再計算する必要がある．これは，非常に大きな計算コストが必要である．文献 [3], [6] では，Pub/Sub 環境においてムービングクエリの解をモニタリングする問題に取り組んでいる．例えば，文献 [3] では，新たに発生したデータが Safe Region に影響を与える可能性のある領域である Impact Region を用いて，クエリの解をモニタリングする手法を提案している．しかし，これらの研究はレンジクエリを対象としており，Top-k クエリを対象とした本研究とは問題が異なる．

6. ま と め

近年，Pub/Sub モデルに基づくアプリケーションやムービングクエリへの関心が高まっている．本稿では，Pub/Sub 環境において，移動するユーザ（ムービングクエリ）に対する Top-k データをモニタリングする問題に取り組んだ．各クエリの Top-k データを効率的にモニタリングするため，クエリを Safe Regin に基づいて管理し，新たに生成されたデータにより，Top-k データが変化する可能性のあるクエリを効率的に限定する新たな四分木（SQ-tree）を提案した．また，データが削除されたとき，または，ユーザが Safe Region の外に出たときに，新たに Top-k データとなるデータを効率的に検索するアルゴリズムを提案した．さらに，近似的な Safe Region を高速に計算する手法を提案した．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)，(A)(18H04095)，および基盤研究 (B)(T17KT0082a) 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] Chen, L., Cong, G., Cao, X., Tan, K.L.: Temporal spatial-keyword top-k publish/subscribe. In: ICDE. pp. 255–266 (2015)
[2] Chen, L., Cong, G., Jensen, C.S., Wu, D.: Spatial keyword query processing: an experimental evaluation. PVLDB 6(3), 217–228 (2013)
[3] Guo, L., Zhang, D., Li, G., Tan, K.L., Bao, Z.: Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In: SIGMOD. pp. 843–857 (2015)
[4] Huang, W., Li, G., Tan, K.L., Feng, J.: Efficient safe-region construction for moving top-k spatial keyword queries. In: CIKM. pp. 932–941 (2012)
[5] Wang, X., Zhang, W., Zhang, Y., Lin, X., Huang, Z.: Top-k spatial-keyword publish/subscribe over sliding window. The VLDB Journal 26(3), 301–326 (2017)
[6] Wang, X., Zhang, Y., Zhang, W., Lin, X., Wang, W.: Ap-tree: efficiently support location-aware publish/subscribe. The VLDB Journal 24(6), 823–848 (2015)
[7] Wu, D., Yiu, M.L., Jensen, C.S., Cong, G.: Efficient continuously moving top-k spatial keyword query processing. In: ICDE. pp. 541–552 (2011)
[8] Žalik, B.: Two efficient algorithms for determining intersection points between simple polygons. Computers & Geosciences 26(2), 137–151 (2000)
[9] Zhang, C., Zhang, Y., Zhang, W., Lin, X.: Inverted linear quadtree: Efficient top k spatial keyword search. TKDE 28(7), 1706–1721 (2016)
[10] 西尾俊哉, 天方大地, 原 隆浩: 位置・キーワードに基づくムービング top-k パブリッシュ/サブスクライブ. 第 10 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 2018）(2018)