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

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．ユーザが移動するたび，そのユーザの Top-k データを再評価し，また，データが発生するたび，全てのクエリの 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，ユーザの移動やデータの発生により 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．

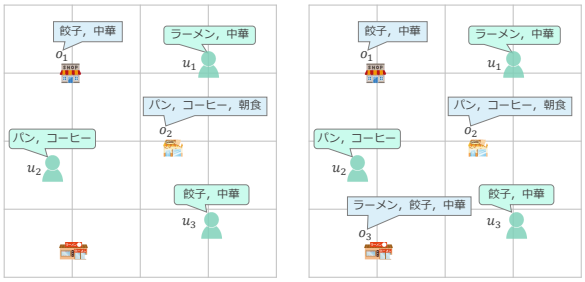
キーワード　Pub/Sub，Top-k クエリ，ムービングクエリ

1. はじめに

近年，GPS を搭載した端末の普及に伴い，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 [4], [7]．これらのアプリケーションの多くは，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．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]．一例として，指定した位置の近くの PoI（Point of Interest）が発信する情報の中で，自身の興味に合うものを受信するアプリケーションが考えられる．

例 1. 図 1 は，Pub/Sub 環境において，3 人のユーザがレストランが発信するデータをモニタリングする例を表している．各ユーザがある位置とキーワードを登録しており，興味に合うデータをモニタリングしている．図 1a において，$k = 1$ であると仮定し，ユーザ u_1 に注目する．u_1 は，ラーメンおよび中華に興味があるため，指定した位置に近く，キーワードが一致するデータ o_1 が Top-k データとなる．同様に，u_2 に対する Top-k データは o_2 であり，u_3 に対する Top-k データは o_1 である．さらに，図 1b において，新たに o_3 が生成された例を示す．このとき，u_3 に注目すると，指定した位置の近くに，キーワードが一致するデータが発生したため，Top-k データが変化する．これにより，各ユーザは自身の興味に最も合うデータをモニタリングできる．

これまでに，位置とキーワードに基づく Top-k パブリッシュ/サブスクライブに関する研究が数多く行われている [1], [6]．これらの研究は，クエリが指定する位置は固定であることを想定しており，ユーザが移動することを考慮していない．しかし，例 1 のようなアプリケーションでは，ユーザは指定した位置に近いデータではなく，現在地に近いデータを要求する [3]．例えば，観光客が歩きながらレストランを探しているとき，近くのレストランで発信される最新の情報やクーポンを受信するといったことが考えられる．本稿では，Pub/Sub モデルで生成されたデータに対して，各ユーザが移動しながら位置およびキーワードに基づく Top-k データをモニタリングする問題に取り組む．



(a) 時刻 t_{cur}　　(b) 時刻 $t_{cur} + 1$

図 1: Pub/Sub 環境における Top-k データモニタリングの一例

課題. ユーザが移動すると，そのユーザに対する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．例えば，図 1b において，u_1 が o_3 の近くに移動した場合，o_1 よりも o_3 のスコアの方が良くなるため，Top-k データが変化する．このような環境において，既存研究におけるアルゴリズムを単純に適用した場合，ユーザが移動するたびに新たにクエ

リを再発行し，その都度 Top-k データの再計算を行う必要がある．また，新しくデータが生成されたとき，ユーザの位置が変化している可能性があるため，すべてのユーザの Top-k データを再計算する必要がある．これは，多くのユーザが存在している環境では多大な時間がかかってしまう．その結果，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．

提案アルゴリズムの概要．これらの課題を解決するため，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 Top-k データのみを更新するアルゴリズムを提案する．ユーザの移動に対しては，文献 [5] において提案された SR（Safe Region) を用いる．SR は，ユーザが移動しても Top-k データが変化しない領域であり，SR を用いることで，ユーザが移動したときに Top-k データが変化する場合を把握できる．新たに生成されたデータに対しては，クエリを SR に基づく木構造で管理し，Top-k データが変化する可能性のあるクエリを限定する．ここで，データのスコアはクエリとデータとの距離，およびキーワードの類似度に基づいて計算される．そのため，クエリを SR に基づいた木構造で管理した場合，クエリの位置が SR 内のどこであるか不明であるため，スコアを正確に計算できない．本稿では，この問題を解決し，クエリが SR 内のどこに存在しても，正確に枝刈りを行う新たなフィルタリング技術を提案する．また，新たに生成されたデータによって SR が変化する場合，インクリメンタルに SR を更新するアルゴリズムを提案する．これにより，各ユーザの Top-k データを効率的にモニタリングする．

貢献と構成．以下に，本研究の貢献を示す．(i) 本稿では，Pub/Sub モデルにおいて，各ユーザが移動しながら Top-k データをモニタリングする問題に取り組む（2. 章）．筆者らの知る限り，この問題はこれまでに取り組まれていない．(ii) クエリを SR に基づく木構造で管理し，ユーザが移動して SR の外に出た場合，または，新たに発生したデータによって Top-k データが変化する可能性のある場合のみ，Top-k データの更新を行う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有効性を確認する（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．これまでに生成されたデータの集合を O とする．あるデータ $o \in O$ は，位置（$o.s$），およびキーワード集合（$o.t$）で構成される（$o = \langle s, t \rangle$）．$o.s$ は，緯度と経度によって表される 2 次元平面上の点とし，時間により変化しないとする．

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-$\boldsymbol{k}$ **S**patial **K**eyword）クエリと呼び，発行されたすべての MkSK クエリの集合を Q とする．

定義 1 (MkSK クエリ)．ある MkSK クエリ $q \in Q$ は，位置（$q.s$），キーワード集合（$q.t$），上位何個のデータを要求するかを指定する変数（$q.k$），および位置とキーワードに対する重みを指定する変数（$q.\alpha$）で構成される（$q = \langle s, t, k, \alpha \rangle$）．$q.t$ は，t_{max} 個以下のキーワードを指定する．O が与えられたとき，q は，スコアの優れた上位 k 個のデータ集合 $A \in O$ を出力する．ここで，A は以下の条件を満たす．

$$\forall o_i \in A, \forall o_j \in O \backslash A, score(q, o_i) \leqq score(q, o_j), |A| = k$$

ここで，$score(q, o)$ は，q に対するデータ o のスコアであり，クエリの現在地に基づいて計算される．

今後，特に明記する必要がない場合，$q.k$，および $q.\alpha$ を単に k，α と表記する．

スコアリング関数．$score(q, o)$ は，式 (1) により定義され，スコアが小さいほど優れているものとする．

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

ここで，$dist(q.s, o.s)$ 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），$text(q.t, o.t)$ 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．距離スコアはユークリッド距離を用いて計算され，$dist(q.s, o.s) = \frac{Edist(q.s, o.s)}{Maxdist}$ である．ここで，$Edist(q.s, o.s)$ 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，$text(q.t, o.t) = \frac{|q.t \cap o.t|}{|q.t \cup o.t|}$ である．

問題定義．O および Q が与えられたとする．本問題は，Q に含まれるすべての q の Top-k データをモニタリングすることである．

2.2 既存研究のアルゴリズム

文献 [5] 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 SR（Safe Region）を作成するアルゴリズムが提案されている．まず，SR および DR（Dominant region）を定義する．

定義 2 (SR)．O，q，および A が与えられたとする．このとき，q の SR（R）は次の条件を満たす．

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

定義 3 (DR)．q 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 DR（D_{o_i, o_j}）は次の条件を満たす．

$$D_{o_i, o_j} = \{s' | q.s = s', score(q, o_i) \leqq score(q, o_j)\}$$

SR および DR の定義から，以下の定理が成り立つ．

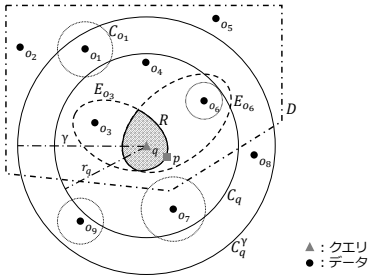


図 2: ASR の例

定理 1. O，q，および A が与えられたとする．このとき，$R = \cap_{o_i \in A} (\cap_{o_j \in O-A} D_{o_i,o_j})$ である．

文献 [5] では SR を高速に計算するため，近似的な SR（ASR）を計算する手法を提案している．提案アルゴリズムは ASR を使用している．ここで，新たに発生したデータに対して ASR をインクリメンタルに更新するアルゴリズムを検討するため，ASR の計算方法について簡単に説明する．

ASR の計算は 3 ステップで行われる．まず，$o_i \in A$ に対して，以下の条件を満たす楕円領域 E_{o_i} を計算し，それらの共通部分 $E = \cap_{o_i \in A} E_{o_i}$ を計算する．

$$E_{o_i} = \{q.s' | dist(q.s', o_i) + dist(q.s, q.s') \leqq \gamma - r_{o_i}\} \quad (2)$$

ここで，$r_{o_i} = \frac{1-\alpha}{\alpha}(1 - text(q.t, o_i.t))$，および $\gamma = \max\{dist(q, o_i) + r_{o_i} | o_i \in A\} + \Delta$ であり，Δ は近似率を決めるパラメータである．また，E_{o_i} は $q.s$ と o_i を焦点とする楕円領域を表す．例えば，図 2 において，$k = 2$ であり，o_3 および o_6 が Top-k データであると仮定する．このとき，それぞれに対する楕円領域 E_{o_3} および E_{o_6} を計算し，それらの共通部分を計算する．次に，$q.s$ を中心とし，半径 r_q 内の領域を C_q，および半径 γ 内の領域を C_q^γ とする．また，各データ o_j に対して，$o_j.s$ を中心とし，半径 r_{o_j} 内の領域を C_{o_j} をする．そして，$C_q^\gamma \backslash C_q$ 内に C_{o_j} が含まれるデータ o_j に対して，D_{o_i,o_j} を計算し，それらの共通部分 $D = \cap_{o_i} (\cap_{o_j \in C_q^\gamma \backslash C_q} D_{o_i,o_j})$ を計算する．ここで，$C_q^\gamma \backslash C_q$ 内に C_{o_j} が含まれるデータとは，$C_{o_j} \notin C_q$ かつ $C_{o_j} \in C_q^\gamma$ を満たすデータを指す．図 2 において，$C_a^\gamma \backslash C_q$ 内に C_{o_j} が含まれるデータとは，例えば o_7 や o_9 である．最後に，E と D の共通部分を計算する（$q.R = E \cap D$）．図 2 の場合，影付きの領域が $q.R$ である．

3. 提案アルゴリズム

提案アルゴリズムは，ユーザが移動した際，または新たにデータが生成された際，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 データを更新する．具体的には，SR を用いて，ユーザが移動したときに Top-k データが変化する場合を把握する．また，クエリを SR に基づく木構造で管理し，Top-k データが変化する可能性のあるクエリを限定する．このとき，SR の重心をクエリの位置として，重心が含まれるノードでクエリを管理する．ここで，SR がノードの領域をはみ出ている場合，クエリがノード内にいることが保証できない．このような場合において，Top-k データの更新が起こり得ないクエリを正確に枝刈りできる新たなフィルタリング技術を提案する．さらに，新たに生成されたデータによって SR が変化する場合，インクリメンタルに SR を更新するアルゴリズムを提案する．表 1 は，本稿で用いる記号の概要を示す．

表 1: 記号の概要

記号	意味
o	データ
O	これまでに生成されたデータの集合
q	MkSK クエリ
Q	MkSK クエリの集合
w	キーワード
R	SR
D	DR
T_{com}	$q.t$ と $o.t$ の間で一致するキーワード集合
$\lambda[i]$	$\lvert T_{com}\rvert = i$ ごとの距離スコアの閾値
n	四分木のノード
Q_n	n を根とする部分木に含まれるクエリの集合
$QList$	中継ノードで管理するクエリの集合
$\Lambda[i]$	Q_n に対する $\lvert T_{com}\rvert = i$ ごとの距離スコアの閾値
$\Lambda_{QL}[i]$	$QList$ に対する $\lvert T_{com}\rvert = i$ ごとの距離スコアの閾値

まず，3.1 節において，Top-k データが変化する可能性のあるクエリを限定する新たなフィルタリング技術を紹介し，3.2 節において，フィルタリング技術を用いて，効率的にクエリを限定するインデックスを紹介する．そして，3.3 節において，Top-k データ，SR，およびインデックスを更新するアルゴリズムを紹介する．

3.1 フィルタリング技術

あるクエリ q の k 番目のデータのスコアを $score_k$ とする．このとき，データ o により，q の Top-k データが更新されるためには，$score_k > score(q, o)$ 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，$q.t$ と $o.t$ の間で一致するキーワード集合（T_{com}）のサイズが i であるとき，キーワードスコアを $text[i]$ と表す．$text[i]$ の上界値を $text_{UB}[i]$ とすると，$text_{UB}[i]$ は以下で与えられる．

補助定理 1. $text_{UB}[i] = \frac{i}{|q.t|}$

証明. $q.t$ と $o.t$ の間で一致するキーワードの数が i 個であるとき，最もキーワードスコアが大きくなるのは，$|o.t| = i$ を満たすときである．$text[i] = \frac{i}{|q.t|+|o.t|-i}$ より，$text_{UB}[i] = \frac{i}{|q.t|}$ である． □

$text_{UB}[i]$ および式（1）から，q の Top-k データが更新されるための距離スコアの閾値 $\lambda_d[i]$ が計算できる．

$$\lambda_d[i] = \frac{score_k}{\alpha} - \frac{1-\alpha}{\alpha}(1 - text_{UB}[i]) \quad (3)$$

定理 2. $\lambda_d[i] < dist(q.s, o.s)$ ならば，o は q の Top-k データになり得ない．

証明. o が q の Top-k データとなるためには，$score_k > score(q,o)$ を満たす必要がある．しかし，$\lambda_d[i] < dist(q.s,o.s)$ ならば，$score_k < \alpha \cdot dist(q.s,o.s) + (1-\alpha)(1-text_{UB}[i])$ である．また，$text_{UB}[i] \geqq text(q.t,o.t)$ より，$\alpha \cdot dist(q.s,o.s) + (1-\alpha)(1-text_{UB}[i]) \leqq \alpha \cdot dist(q.s,o.s) + (1-\alpha)(1-text(q.t,o.t)) = score(q,o)$ であるため，$\lambda_d[i] < dist(q.s,o.s)$ ならば，$score_k < score(q,o)$ である． □

定理 2 より，$\lambda_d[i] < dist(q.s,o.s)$ ならば，正確性を失うことなく q を枝刈りできる．

提案アルゴリズムでは，クエリを SR に基づいて管理するが，クエリの位置は R 内のどこであるか不明である．このとき，クエリの位置により $score_k$ が変化し，$\lambda_d[i]$ が変化するため，このままでは正確に枝刈りできない．そこで，式（3）を拡張し，クエリが R 内のどこに存在しても，正確な枝刈りが可能な閾値を導出する．まず，クエリが R 内の任意の位置にいる場合の $score_k$ の上界値 $score_{UB}$ を考える．

補助定理 2. $score_{UB} = \frac{\gamma \cdot \alpha + score_k}{2}$

証明. R は，q の各 Top-k データ o_i に対する E_{o_i} に必ず含まれる．さらに，q が E_{o_i} 内の任意の位置にいるとき，$score(q,o_i)$ の上界値 $score_{UB}(q,o_i)$ を考える．図 2 における E_{o_3} に注目する．クエリが点 p にいるとき最もスコアが大きくなるため，

$$\begin{aligned} &score_{UB}(q,o_3) \\ &\quad = \alpha \cdot dist(p,o_3) + (1-\alpha)(1-text(q.t,o_3.t)) \\ &\quad = \alpha \cdot (dist(p,o_3) + r_{o_3}) \end{aligned}$$

である．また，$dist(p,o_3) = \frac{\gamma - r_{o_3} + dist(q,o_3)}{2}$ で与えられるため，

$$\begin{aligned} score_{UB}(q,o_3) &= \alpha \cdot \frac{\gamma + r_{o_3} + dist(q,o_3)}{2} \\ &= \frac{\gamma \cdot \alpha + score(q,o_3)}{2} \end{aligned}$$

である．$score_{UB} = \max\{score_{UB}(q,o_i) | o_i \in A\}$ であるため，補助定理 2 が成り立つ． □

式（3）および補助定理 2 から，クエリが R 内の任意の位置にいるとき，Top-k データが更新されるための距離スコアの閾値 $\lambda[i]$ が計算できる．

$$\lambda[i] = \frac{score_{UB}}{\alpha} - \frac{1-\alpha}{\alpha}(1 - text_{UB}[i]) \qquad (4)$$

ここで，$o.s$ から R までの距離スコアの最小値を $dist(R,o.s)$ とすると，以下の定理が成り立つ．

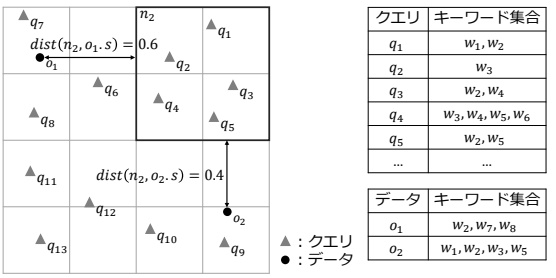
定理 3. $\lambda[i] < dist(R,o.s)$ ならば，o は q の Top-k データになり得ない．

証明. 定理 2 と同様． □

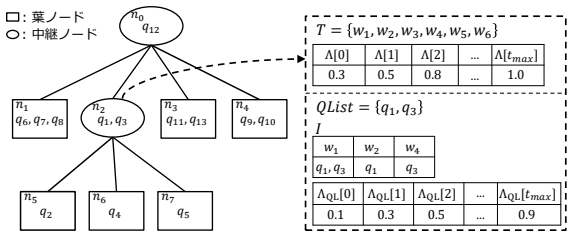
さらに，定理 3 から，以下の系が成り立つ．

系 1. $\lambda[i] < dist(R,o.s)$ ならば，o により R が変化することはない．

定理 3 および系 1 より，$\lambda[i] < dist(R,o.s)$ ならば，正確性を失うことなく q を枝刈りでき，R を更新する必要もない．



(a) クエリおよびデータの位置とキーワード



(b) 四分木および n_2 に保存される情報

図 3: 四分木の例

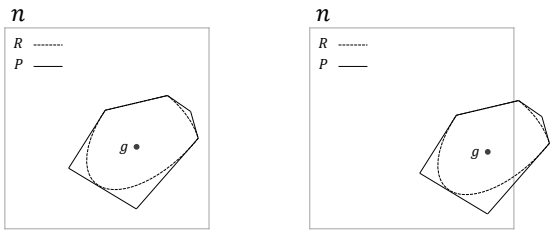
3.2 クエリのインデックス

提案アルゴリズムでは，3.1 節で紹介したフィルタリング技術を用いて，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ため，クエリを SR に基づいて四分木により管理する．四分木を構築する時点で図 3a に示すようにクエリが存在していると仮定すると，図 3b の左図に示す四分木が構築される．提案アルゴリズムにおける四分木は，葉ノードだけでなく，中継ノードでもクエリを管理し，各ノードで管理するクエリの集合を $QList$ とする．

また，各ノードは，そのノードを根とする部分木に含まれるクエリのキーワード集合 T，および $|T_{com}| = i$ ごとの距離スコアの閾値 $\Lambda[i]$ を保存している．さらに，$QList \neq \emptyset$ である場合，$QList$ に含まれるクエリごとにクエリへのポインタを管理する転置ファイル I，および $|T_{com}| = i$ ごとの距離スコアの閾値 $\Lambda_{QL}[i]$ を保存する．$\Lambda[i]$ および $\Lambda_{QL}[i]$ の詳細は後述する．例えば，図 3b の右図はノード n_2 に保存される情報を示している．n_2 を根とする部分木には，クエリ $q_1, q_2, q_3, q_4,$ および q_5 が含まれるため，それらのキーワード集合および $\Lambda[i]$ を保存している．また，$n_2.QList$ には q_1 および q_3 が含まれるため，それらのキーワード集合の転置ファイルおよび $\Lambda_{QL}[i]$ を保存している．

まず，四分木の構築方法を紹介し，次に，ノードに保存された情報を用いたフィルタリング方法を紹介する．

四分木の構築. クエリ q の位置は，R 内のどこであるか不明であるため，四分木により管理することが単純にはできない．そこで，まず始めに，R を包含する多角形 P を生成し，P の重心 g を計算する．g を q の位置として，g が含まれるノードで q を管理する．次に，q をあるノード n で管理することを考える．このとき，図 4 に示すように，P が n に含まれる場合（図 4a），および P が n からはみ出る場合（図 4b）の 2 通りが考えられる．ここで，$o.s$ から n までの距離スコアの最小値



(a) $P \in n$ (b) $P \notin n$

図 4: q を n で管理するときの例

を $dist(n, o.s)$ とし，$o.s$ が n に含まれる場合，$dist(n, o.s) = 0$ とする．詳細は後述するが，提案アルゴリズムでは，あるデータ o が発生したとき，$dist(n, o.s) \leqq dist(R, o.s)$ が成り立つことに基づいてフィルタリングを行う．しかし，図 4b の場合，$dist(n, o.s) \leqq dist(R, o.s)$ が成り立つとは限らない．そこで，P が n からどれだけはみ出ているかを表す距離スコアを $dist_{add}(P, n)$ とすると，$dist_{add}(P, n)$ は式（5）で与えられる．

$$dist_{add}(P, n) = \max(0, dist_{max}(g, P) - dist_{min}(g, n)) \quad (5)$$

ここで，$dist_{max}(g, P)$ は，g と P の各頂点の間の距離スコアの最大値，$dist_{min}(g, n)$ は，g から n の境界までの距離スコアの最小値である．このとき，以下の補助定理が成り立つ．

補助定理 3. $dist(n, o.s)$，$dist(R, o.s)$，および $dist_{add}(P, n)$ が与えられたとき，$dist(n, o.s) \leqq dist(R, o.s) + dist_{add}(P, n)$ が成り立つ．

$dist_{add}(P, n)$ を考慮し，$\Lambda[i]$ および $\Lambda_{QL}[i]$ は，それぞれ以下の式で与えられる．

$$\Lambda[i] = \max\{q.\lambda[i] + dist_{add}(q.P, n) | q \in Q_n\} \quad (6)$$

$$\Lambda_{QL}[i] = \max\{q.\lambda[i] + dist_{add}(q.P, n) | q \in n.QList\} \quad (7)$$

ここで，Q_n は n を根とする部分木に含まれるクエリの集合である．また，$|q.t| < i$ のとき，$q.\lambda[i] = q.\lambda[|q.t|]$ とする．詳細は後述するが，$dist_{add}(P, n)$ を考慮し，$\Lambda[i]$ および $\Lambda_{QL}[i]$ を計算することで，図 4b のような場合において正確性を失うことなくフィルタリングを行うことができる．

次に，クエリを管理するノードを決定する方法を説明する．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割することが望ましい．しかし，四分木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，キーワードに関して効率的に分割を行うことが困難である．提案アルゴリズムでは，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．

ここから，クエリをどのノードで管理するかを決定するアルゴリズムについて紹介する．まず，クエリ q をあるノード n で管理すると仮定し，あるデータ o が発生したとき，n に対して計算される $dist(n, o.s)$ の期待値を $dist_e(n)$[(注1)] とする．このとき，式（1）および補助定理 2 から，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 $text_e$ が計算される．

$$text_e = 1 - \frac{score_{UB} - \alpha \cdot dist_e(n)}{1 - \alpha} \quad (8)$$

$|T_{com}| = i$ のとき，$text_{UB}[i] = \frac{i}{|q.t|}$ であるため，q の Top-k データが更新されるために必要な T_{com} の数の期待値 $q.i_e$ は，以下の式で与えられる．

$$q.i_e = \lfloor text_e \cdot |q.t| \rfloor \quad (9)$$

また，過去に発生したデータのキーワードの特徴に基づき，新たに発生したデータのキーワードと $n.T$ の間で一致するキーワードの数の期待値が $n.i_e$ であると仮定する．このとき，$n.i_e < q.i_e$ ならば，新たに発生したデータに対して，q を枝刈りできる可能性が大きい．そのため，このようなクエリは，子ノードへ分配せず n で管理する．

上で紹介したアルゴリズムを基に，新たに発生したクエリ q を四分木に挿入するアルゴリズムを紹介する（アルゴリズム 1）．g が n に含まれる場合，まず，T，$n.i_e$，および $\Lambda[i]$ を更新し，$dist_{add}(P, n)$ を計算する（2–7 行）．次に，n が葉ノードである場合，$QList$ および I を更新する（9 行）．その後，$|QList| > m$ である場合，4 つの子ノードを作成し，$n.i_e \geqq q'.i_e$（$q' \in QList$）を満たすクエリを子ノードに分配し，分配されなかったクエリに対して，$n.\Lambda_{QL}[i]$ を計算するアルゴリズム Subdivide を実行する（11 行）．ここで，m は，葉

Algorithm 1: Insert(q, n)

```
   Input: q, n
1  if g ∈ n then
2      T ← T ∪ q.t
3      Update n.i_e
4      Calculate dist_add(P, n)
5      for 0 ≦ i ≦ t_max do
6          if Λ[i] < q.λ[i] + dist_add(P, n) then
7              Λ[i] ← q.λ[i] + dist_add(P, n)
8      if n is a leaf node then
9          QList ← QList ∪ {q}, I ← I ∪ q.t
10         if |QList| > m //m is node capacity then
11             Subdivide
12     else
13         Disribute
14         Calculate q.i_e
15         if n.i_e < q.i_e then
16             QList ← QList ∪ {q}, I ← I ∪ q.t
17             for 0 ≦ i ≦ t_max do
18                 if Λ_QL[i] < q.λ[i] + dist_add(P, n) then
19                     Λ_QL[i] ← q.λ[i] + dist_add(P, n)
20         else
21             for ∀n_c ∈ N //N is n's children do
22                 Insert(q, n_c)
```

(注1)：$dist_e(n) = \frac{1}{S} \iint_{\mathbb{R}^2} dist(n, o) dx dy$ （S は領域 $\mathbb{R}^2$ の面積である）

Algorithm 2: NodeFiltering(o, n_c)

Input: o and n// o is a new generated data

1 $Q_u \leftarrow \emptyset$
2 $i_c \leftarrow$ the number of common keywords between T and $o.t$
3 **if** $i_c \geqq t_{max}$ **then**
4 $\quad i_c \leftarrow t_{max}$
5 Calculate $dist(n, o.s)$
6 **if** $\Lambda[i_c] \geqq dist(n, o.s)$ **then**
7 $\quad$ **if** *n is not a leaf node* **then**
8 $\quad\quad$ **for** $\forall n_c \in N$ *//N is n's children* **do**
9 $\quad\quad\quad Q_c \leftarrow$NodeFiltering$(o, n_c)$
10 $\quad\quad\quad Q_u \leftarrow Q_u \cup Q_c$
11 $\quad$ **if** $|QList| > 0$ **then**
12 $\quad\quad Q_{QL} \leftarrow$ QueryFiltering$(o, dist(n, o.s), i_c)$
13 $\quad\quad Q_u \leftarrow Q_u \cup Q_{QL}$
14 return Q_u

ノードで管理するクエリの数を決めるパラメータである．n が中継ノードである場合，$n.i_e$ の更新により，$n.i_e \geqq q'.i_e$ となった q' を子ノードに分配するアルゴリズム Distribute を実行する（13 行）．次に，$q.i_e$ を計算し，$n.i_e < q.i_e$ ならば，$QList$，I，および $\Lambda_{QL}[i]$ を更新する（16–19 行）．$n.i_e \geqq q.i_e$ ならば，子ノードに再帰的に挿入する（21–22 行）．

フィルタリング. 提案アルゴリズムにおけるフィルタリングでは，ノードのフィルタリング，およびクエリのフィルタリングの 2 段階のフィルタリングを行う．データ o が生成され，あるノード n をチェックしたと仮定し，それぞれのフィルタリングについて説明する．

まず，ノードのフィルタリングについて説明する．アルゴリズム 2 は，ノードのフィルタリングを行うアルゴリズムを示している．まず，T と $o.t$ の間で一致するキーワードの数 i_c を計算し，$i_c \geqq t_{max}$ ならば $i_c = t_{max}$ とする（2–4 行）．このとき，以下の定理が成り立つ．

定理 4. $\Lambda[i_c] < dist(n, o.s)$ ならば，o は $q \quad \in Q_n$ の Top-k データになり得ない．

証明. $q.t$ と $o.t$ の間で一致するキーワードの数を j 個とすると，明らかに $j \leqq i_c$ であり，$q.\lambda[i]$ は i の増加に伴い，単調に増加するため，$q.\lambda[j] \leqq q.\lambda[i_c]$ である．また，$q.\lambda[i_c] + dist_{add}(q.P, n) \leqq \Lambda[i_c]$ であるため，$q.\lambda[j] + dist_{add}(q.P, n) < dist(n, o_s)$ である．補助定理 3 より，$q.\lambda[j] + dist_{add}(q.P, n) < dist(R, o.s) + dist_{add}(q.P, n)$ であるから，$q.\lambda[j] < dist(R, o.s)$ が成り立つ．よって，定理 3 より，定理 4 が証明される． □

定理 4 より，$\Lambda[i_c] < dist(n, o.s)$ ならば，正確性を失うことなく，Q_n に含まれるすべてのクエリを枝刈りでき，それぞれの R の更新も考えなくてよい．次に，ノードのフィルタリングを行い，枝刈りされなかったノードに対して，子ノードが存在する場合，子ノードに対して，ノードのフィルタリングを再帰的に行い，また，$QList$ が存在する場合，$QList$ に含まれるクエリに対して，クエリのフィルタリングを行う（6–13 行）．

Algorithm 3: QueryFiltering$(o,\ dist(n, o.s),\ i_c)$

Input: n, o, $dist(n, o.s)$, and i_c
Output: Q_{QL}

1 $Q_{QL} \leftarrow \emptyset$
2 **if** *n is a leaf node* **then**
3 $\quad i_{min} \leftarrow \min\{i | \Lambda[i] \geqq dist(n, o.s)\}$
4 **else**
5 $\quad i_{min} \leftarrow \min\{i | \Lambda_{QL}[i] \geqq dist(n, o.s)\}$
6 **if** $i_{min} \leqq i_c$ **then**
7 $\quad$ **for** $\forall q_j \in QList$ *where* $|q_j.t \cap o.t| \geqq i_{min}$ **do**
8 $\quad\quad Q_{QL} \leftarrow Q_{QL} \cup q_j$
9 return Q_{QL}

例 2. 図 3 において，データ o_1 が発生し，n_2 に対してノードのフィルタリングを行う．T と $o_1.t$ の間で一致するのは w_2 のみであるため $i_c = 1$ である．$dist(n_2, o_1.s) = 0.6$ より，$\Lambda[i_c] < dist(n_2, o_1.s)$ であるため，n_2 以下に含まれるすべてのクエリは枝刈りされ，Top-k データおよび SR の更新を行わない．

ここから，クエリのフィルタリングについて説明する．アルゴリズム 3 は，クエリのフィルタリングを行うアルゴリズムを示している．まず，$\Lambda_{QL}[i] \geqq dist(n, o.s)$ を満たす最小の i を i_{min} とする（2 行）．このとき，以下の定理が成り立つ．

定理 5. $q.t$（$q \in QList$）と $o.t$ の間で一致するキーワードの数を j 個とすると，o により，Top-k データが変化する可能性のあるクエリは $i_{min} \leqq j$ を満たすクエリのみである．

証明. $j < i_{min}$ であると仮定すると，$\Lambda_{QL}[j] < dist(n, o.s)$ である．このとき，定理 4 の証明と同様に，$q.\lambda[j] < dist(R, o.s)$ が成り立つため，o は $j < i_{min}$ であるような q の Top-k データになり得ない．よって，定理 5 が証明される． □

定理 5 により，正確性を失うことなく，一致するキーワードの数が i_{min} に満たないクエリを枝刈りでき，それぞれの R の更新も考えなくてよい．また，NodeFiltering の際に計算された i_c よりも i_{min} の方が大きいとき，$QList$ に含まれるクエリの中で，一致するキーワードの数が i_{min} 以上になるものは存在しない．そのため，$i_{min} \geqq i_c$ を満たす場合のみ，転置ファイル I を用いて，各クエリと $o.t$ の間で一致するキーワードの数を計算し，i_{min} 以上となるクエリのみ，更新候補に加える（3–5 行）．ノードのフィルタリング，およびクエリのフィルタリングを再帰的に行うことにより，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ことができる．

例 3. 図 3 において，データ o_2 が発生し，$n_2.QList$ に対してクエリのフィルタリングを行う．$dist(n_2, o_2.s) = 0.4$ より，$i_{min} = 2$ である．ここで，I を用いて，$o_2.t$ との間で一致する

キーワードの数を計算すると，q_1 は 2 個，q_3 は 1 個である．そのため，クエリのフィルタリングの結果，q_1 のみを更新候補に加える．

3.3 更新アルゴリズム

本節では，フィルタリングにより限定されたクエリの Top-k データおよび SR を更新した後，インデックスを更新する方法を説明する．

Top-k データの更新．まず，ユーザに現在地を要求し，クエリの位置を更新した後，各 Top-k データに対するスコアを更新する．その後，発生したデータのスコアを計算し，Top-k データを更新する．

SR の更新．Top-k データの更新後，Top-k データとならなかったデータを o とする．ユーザが，R 内の任意の位置にいるとき，o'（$o' \in O - A - \{o\}$）は Top-k データになり得ない．そこで，提案アルゴリズムでは，R 内で o が Top-k データとなる領域を取り除くことで，SR の更新を実現する．

具体的には，各 Top-k データ o_i に対して，$D_{o_i,o}$ を計算し，それらの共通部分 $D_o = \cap_{o_i \in A} D_{o_i,o}$ を計算する．D_o は，o が Top-k データとなる領域を表す．次に，2.2 節で紹介したアルゴリズムを用いて SR を計算した時点の $q.s$ を $q.s_o$ としたとき，$q.s_o$ が D_o に含まれるかどうかをチェックする．D_o に含まれる場合，$R \cap D_o$ を新たな SR とする．D_o に含まれない場合，2.2 節で紹介したアルゴリズムを用いて，SR を再計算する．

ここで，既存研究における SR は，$q.s_o$ を原点とする，極形式で SR を管理しており，基準線と成す角度に対して一意に SR を管理する．そして，ユーザが移動したとき，現在地と原点との距離，および現在地と原点を結ぶ線分が基準線と成す角度に基づいて，SR の外に出たかどうかをチェックする．しかし，$q.s_o$ が D_o に含まれない場合，$R \cap D_o$ を $q.s_o$ を原点とする極形式で一意に管理することができない．その結果，ユーザが SR の外に出たかどうかをチェックすることが困難になる．また，SR の一部は，$q.s_o$ を焦点とする楕円曲線で表現されるため，その他の点を原点とする極形式で表すことが困難である．以上の理由から，提案アルゴリズムでは，$q.s_o$ が D_o に含まれない場合，SR を再計算する．

クエリインデックスの更新．クエリの Top-k データおよび SR が更新されると，四分木に保存された情報を更新する必要がある．クエリ q の Top-k データおよび SR が更新されると，更新後の SR の重心 g_{new} を計算する．そして，g_{new} が q を管理するノードに含まれる場合，そのノードおよび上位のノードに保存された情報を更新する．一方，ノードに含まれない場合，q を一旦，四分木から取り除き，再度，更新後の SR に基づいて四分木に挿入する．この操作を，更新が起きたクエリに対して，同様に行うことにより，四分木の更新を実現する．

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．提案アルゴリズムの性能を評価するため，ベースラインアルゴリズムとの比較を行った．ベースラインアルゴリズムは，2.2 節で紹介したアルゴリズムを用いて SR を計算する．そして，ユーザの移動に対しては，提案アルゴリズムと同様の処理を行う．また，新たにデータが生成されると，すべてのクエリにアクセスしてそのデータのスコアを計算し，必要があれば Top-k データおよび SR の更新を行う．

表 2: パラメータの設定

パラメータ	値
クエリの数，$\|Q\|$ $[\times 10^6]$	0.25, **0.5**, 0.75, 1
$q.k$ の最大値，k_{max}	5, **10**, 15, 20
データの発生頻度，$f(/ts)$	10, **20**, 50, 100

4.1 セッティング

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．

データセット．本実験では，生成されるデータとして Twitter [1] を用いた．各クエリは，1 つのツイート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2) および TYO(注3) を用いた．各トラジェクトリは，1 秒を 1 タイムスタンプ（$1ts$）として，ATC は 100 個の連続する点を含むトラジェクトリを抽出し，TYO は 1,000 個の連続する点を含むトラジェクトリを作成し，ユーザのトラジェクトリとした．

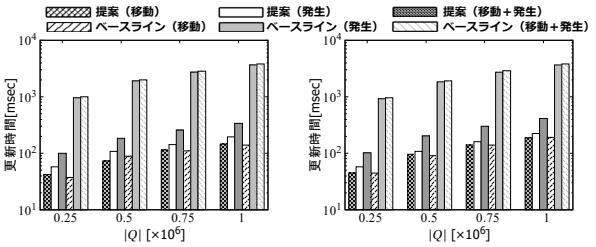
パラメータ．表 2 により，本実験で用いたパラメータを示す．太字で表されている値はデフォルトの値である．また，$t_{max} = 5$ および $m = 1{,}000$ とした．さらに，k は 1 から k_{max} の間の一様乱数とし，α は 0 から 1 の間の一様乱数とした．実験は，データが 1,000,000 個発生した時点から開始する．

4.2 評価結果

本節では，各アルゴリズムにおける平均更新時間（1 タイムスタンプあたりのユーザの移動（移動），およびデータの発生（発生）による 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．

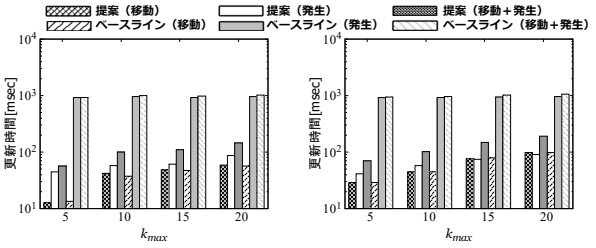
$|Q|$ の影響．図 5 に $|Q|$ を変えたときの結果を示す．$|Q|$ が増加すると，各アルゴリズムにおいて，移動および発生に対する更新時間が大きくなることがわかる．$|Q|$ が大きいとき，ベースラインアルゴリズムは更新時間が 1 秒以上になっており，次のデータが発生するまでに処理を行うことができない．また，発生に対する更新時間において，提案アルゴリズムはベースラインアルゴリズムを大きく上回る性能を示していることがわかる．提案アルゴリズムにおける発生に対する更新時間はベースラインアルゴリズムの 10 分の 1 以下である．これは，ベースラインアルゴリズムがすべてのクエリに対して Top-k データの更新を行うのに対して，提案アルゴリズムは新たに発生したデータにより Top-k データが変化する可能性のあるクエリを限定し，

(注2)：http://www.irc.atr.jp/crest2010_HRI/ATC_dataset/
(注3)：https://sites.google.com/site/yangdingqi/home/foursquare-dataset/



(a) ATC (b) TYO

図 5: $|Q|$ の影響



(a) ATC (b) TYO

図 6: k_{max} の影響

それらの Top-k データのみを更新しているからである．

k_{max} の影響． 図 6 に k_{max} を変えたときの結果を示す．k_{max} が大きくなると，各アルゴリズムにおいて，移動および発生に対する更新時間が大きくなることがわかる．k_{max} が大きくなると，よりスコアの大きいデータが Top-k データとなる．その結果，新たに発生したデータにより Top-k データおよび SR が変化する可能性が大きくなり，発生に対する更新時間が大きくなる．また，Top-k データとなり得るデータの数が増加するため，ユーザが SR の外に出る可能性が大きくなる．さらに，スコアの大きいデータが Top-k データとなると，$C_q^\gamma \backslash C_q$ 内に含まれるデータの数が多くなるため，SR の計算時間が大きくなる．その結果，移動に対する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], [6] 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 [5], [8] 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，2.2 節において紹介した，SR を計算することで，ユーザが SR の外に出た場合のみ Top-k 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 SR を再計算する必要がある．文献 [3], [7] 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．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. 終 わ り に

近年，パブリッシュ/サブスクライブモデルに基づくアプリケーションが多く存在する．また，ユーザが移動しながら，現在地に近いデータを検索するムービングクエリ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，ユーザが移動しながら位置およびキーワードに基づく有用な上位 k 個のデータを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 SR に基づいた木構造で管理し，ユーザの移動やデータの発生によって 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 データの更新を行う．これにより，各ユーザの Top-k データの更新を高速に行う．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．

提案アルゴリズムでは，ユーザが移動し SR の外に出た場合，Top-k データを更新した後，SR を一から再計算する．これをインクリメンタルに更新するアルゴリズムに拡張することが今後の課題として挙げられる．

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)，基盤研究 (B)(T17KT0082a)，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] L. Chen, G. Cong, X. Cao, and K.-L. Tan. Temporal spatial-keyword top-k publish/subscribe. In *ICDE*, pages 255–266, 2015.

[2] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: an experimental evaluation. *PVLDB*, 6(3):217–228, 2013.

[3] L. Guo, D. Zhang, G. Li, K.-L. Tan, and Z. Bao. Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In *SIGMOD*, pages 843–857, 2015.

[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] W. Huang, G. Li, K.-L. Tan, and J. Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *CIKM*, pages 932–941, 2012.

[6] 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.

[7] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. Ap-tree: efficiently support location-aware publish/subscribe. *The VLDB Journal*, 24(6):823–848, 2015.

[8] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552, 2011.