位置およびキーワードに基づく近似逆 Top-k 検索のための バッチ処理アルゴリズム

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，位置情報およびキーワードを含むオブジェクトおよびサブスクリプションが大量に生成されている．本稿では，ART（Approximate Reverse spatio-textual Top-k）クエリ処理問題に取り組む．ART クエリは，オブジェクト集合およびサブスクリプション集合が与えられたとき，クエリオブジェクトを有用な上位 k 個のオブジェクト（Top-k オブジェクト）に含むサブスクリプションに加えて，近似的に Top-k オブジェクトに含むサブスクリプションを検索する．ART クエリの解となり得るすべてのサブスクリプションに対して Top-k オブジェクトを計算する単純な方法は非効率的であり，サブスクリプションが多く存在する環境に対応できない．この問題を解決するため，クエリオブジェクトが Top-k オブジェクトに含まれないと判断されたサブスクリプションに対する処理を途中で中断することで，計算時間を削減する手法を紹介する．また，複数の ART クエリが同時に与えられたとき，それらに対してバッチ処理を行う手法およびマルチコア処理を行う手法を提案する．実データを用いた実験により，提案手法の有効性を示す．

キーワード　逆 Top-k クエリ，近似 Top-k クエリ，バッチ処理，マルチコア処理

1. ま え が き

近年，GPS を搭載した端末や SNS の普及に伴い，位置情報やキーワードを含むオブジェクトが大量に生成されている．これらのオブジェクトは，位置に関連したイベントや広告など，ユーザにとって有用な情報を多く含んでいる．多くのアプリケーションでは，ユーザが自身の興味のある位置やキーワードをサブスクリプションとしてシステムに登録することで，有用な情報を受信する [8], [9]．このような環境において，あるオブジェクトに対して自身に興味を持つサブスクリプション（潜在顧客）を把握することは，市場分析やマーケティング調査を行う上で重要な課題であり，これまでに，位置情報およびキーワードに基づいて潜在顧客を検索する逆 Top-k 検索に関する研究が行われている [6], [12]．また，市場分析などへの応用では，潜在顧客を厳密に把握する必要のない場合も多く存在する [4], [5]．あるサブスクリプション s に対する上位 k 番目のオブジェクト o と $k+1$ 番目のオブジェクト o' のスコアが近い値を取る場合，それらの有用度は同程度であると考えられる．このとき，s は o' に対する逆 Top-k 検索において近似的な解とみなすことができる．このようなサブスクリプションを検索可能になれば，市場分析の効率化が期待できる．そこで本稿では，近似逆 Top-k クエリ（ART クエリ）処理問題に取り組む．

課題．ART クエリの解を求める最も単純な方法は，クエリオブジェクトのキーワードを 1 つでも含むすべてのサブスクリプションに対して Top-k 検索を行うものである．これは，サブスクリプションが大量に存在する環境では多大な計算コストが必要である．そのため，クエリの解を高速に検索するアルゴリズムが必要である．また，逆 Top-k 検索の応用では，ある時刻に同時にクエリが発行されることが考えられる．例えば，複数のレストランが，正午時点での潜在顧客を把握するために，逆 Top-k 検索を行うことが考えられる．このような環境に対応するため，複数のクエリに対してバッチ処理を行い，各クエリの解を高速に求めるアルゴリズムが望ましい．さらに，アプリケーションを利用するユーザの数は今後も増え続け，より膨大なオブジェクトが生成されるようになると予想されるため [2]，マルチコア環境で処理を行うアルゴリズムが望ましい．

貢献．以下に，本稿の貢献を示す．

- ART クエリの集合に対してバッチ処理を行い，各クエリの解を高速に検索するアルゴリズム（B-PART），および B-PART をマルチコア環境で行うアルゴリズム（PB-PART）を提案する（4 章および 5 章）．
- 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（6 章）．

上記の内容に加えて，7 章で関連研究について述べ，8 章で本稿をまとめる．

2. 予 備 知 識

本章では，本稿で考える問題を詳細に定義する．まず，位置およびキーワードに基づくオブジェクトおよびサブスクリプションを定義する．

定義 1 (オブジェクト)．あるオブジェクトは，$o = (l, \mathcal{W})$ と定義される．ここで，$o.l$ は緯度と経度によって表される 2 次元平面上のオブジェクト o の位置，$o.\mathcal{W}$ はオブジェクト o がもつキーワードの集合である．

定義 2 (サブスクリプション)．あるユーザの発行するサブスクリプションは，$s = (l, \mathcal{W})$ と定義される．ここで，$s.l$ はサブスクリプション s の位置，$s.\mathcal{W}$ はサブスクリプション s がもつ $1 \leqq |s.\mathcal{W}| \leqq \mathcal{W}_{max}$ 個のキーワードの集合である．

ここで，オブジェクトの集合を O，サブスクリプションの集合を S とする．あるオブジェクト $o \in O$ があるサブスクリプション $s \in S$ に対して有用であるかどうかを判断するため，s に対して有用な上位 k 個のオブジェクトとして Top-k オブジェクトを以下のように定義する．

定義 3 (Top-k オブジェクト)．オブジェクト集合 O が与えられたとき，あるサブスクリプション $s \in S$ の Top-k オブジェクト T は次の条件を満たす．(i) $T \subseteqq O_{s.\mathcal{W}}$，(ii) $|T| = k$，(iii) $\forall o \in T, \forall o' \in O_{s.\mathcal{W}} \backslash T, Edist(s.l, o.l) < Edist(s.l, o'.l)$．ここで，$O_{s.\mathcal{W}}$ は，O に含まれるオブジェクトの中で $s.\mathcal{W}$ に含まれるキーワードを 1 つ以上含むオブジェクトの集合，$Edist(s.l, o.l)$ は，$s.l$ と $o.l$ とのユークリッド距離である．

次に，位置およびキーワードに基づく逆 Top-k クエリを定義する．以降，RT（**R**everse spatio-textual **T**op-k）クエリと呼ぶ．

定義 4 (RT クエリ)．オブジェクト集合 O およびサブスクリプション集合 S が与えられたとき，RT クエリ $q = \{o_q, k\}$ $(o_q \in O)$ は，クエリオブジェクト o_q を Top-k オブジェクトに含むサブスクリプションを検索する．つまり，q の解であるサブスクリプションの集合 RT は次の条件を満たす．$q.RT = \{s \in S | o_q \in s.T\}$.

最後に，位置およびキーワードに基づく近似逆 Top-k クエリを定義する．以降，ART（**A**pproximate **R**everse spatio-textual **T**op-k）クエリと呼ぶ．

定義 5 (ART クエリ)．オブジェクト集合 O およびサブスクリプション集合 S が与えられたとき，ART クエリ $q = \{o_q, k, \delta\}$ $(o_q \in O, \delta \geqq 1)$ の解であるサブスクリプションの集合を ART とする．RT クエリ $q' = \{o_q, k\}$ の解であるサブスクリプションの集合を RT とすると，各サブスクリプション $s \in S$ は，以下の 3 つの場合に分類される．

（1） $s \in RT$ ならば，$s \in ART$ である．

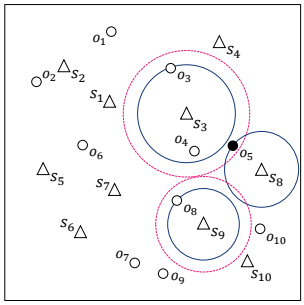
（2） $s \notin RT$ かつ $Edist(s.l, o_q.l) \leqq \delta \cdot Edist(s.l, o_k.l)$ ならば，s は ART に含まれてもよい．

（3） $s \notin RT$ かつ $\delta \cdot Edist(s.l, o_k.l) < Edist(s.l, o_q.l)$ ならば，$s \notin ART$ である．

ここで，o_k は，s に対する上位 k 番目のオブジェクトである．

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

例 1. 図 1 は，本稿を通して用いる例を示している．この例では，10 個のサブスクリプション $\{s_1, \cdots, s_{10}\}$ および 10 個のオブジェクト $\{o_1, \cdots, o_{10}\}$ が存在している．このとき，ART クエリ $q = \{o_5, 1, \delta\}$ が与えられたとする．まず，o_5 は s_8 の Top-1 オブジェクトであるため，s_8 は $q.ART$ に含まれる．次に，s_3 および s_9 の Top-1 オブジェクトはそれぞれ o_3 および o_8 である．しかし，o_5 は $\delta \cdot Edist(s_3.l, o_3.l)$ の範囲内（点線の範囲内）に含まれているため，s_3 は $q.ART$ に含まれてもよい．一方，$\delta \cdot Edist(s_9.l, o_8.l)$ の範囲内には含まれないため，s_9 は $q.ART$ に含まれない．



△: サブスクリプション　○: オブジェクト

s	キーワード
s_1	w_1, w_2
s_2	w_1
s_3	w_2, w_3
s_4	w_1, w_4, w_5, w_6
s_5	w_1, w_5
s_6	w_1, w_6
s_7	w_1, w_4, w_5
s_8	w_2
s_9	w_1, w_3
s_{10}	w_1

o	キーワード
o_1	w_1, w_4
o_2	w_1, w_2, w_3
o_3	w_1, w_2
o_4	w_1, w_4, w_5
o_5	w_2, w_3
o_6	w_3, w_6, w_7
o_7	w_1, w_2
o_8	w_1, w_3
o_9	w_1, w_2, w_3, w_5
o_{10}	w_1, w_7, w_8

図 1: サブスクリプションおよびオブジェクトの位置およびキーワードの例

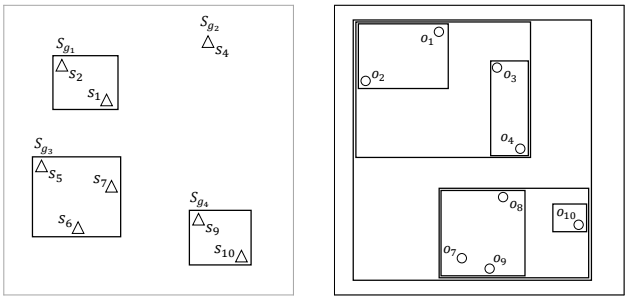
問題定義．オブジェクト集合 O，サブスクリプション集合 S，および ART クエリの集合 Q が与えられたとき，各 ART クエリ $q \in Q$ の解を高速に計算する．

ある ART クエリが与えられたとき，3 章で紹介する方法で解を求めることが可能である．

3. PART

本章では，ART クエリの集合 Q が与えられたとき，各クエリの解を高速に求めるアルゴリズム PART（**P**rocessing of **ART** クエリ）について紹介する．PART は，Q に含まれる各 ART クエリ $q \in Q$ の解を順に計算する．ある ART クエリ $q = (o_q, k, \delta)$ の解を計算する際，解となり得るサブスクリプションは，$o_q.\mathcal{W}$ に含まれるキーワードを少なくとも 1 つ含むすべてのサブスクリプションの集合 $S_{o_q.\mathcal{W}}$ である．ここで，q の解を計算する際，各サブスクリプション $s \in S_{o_q.\mathcal{W}}$ の正確な Top-k オブジェクトを求める必要はなく，o_q が s の Top-k オブジェクトに含まれるかどうかのみ判断できれば良い．PART では，o_q が s の Top-k オブジェクトに含まれないと判断された時点で s のチェックを終了する．具体的には，s の Top-k オブジェクトになり得る $s.\mathcal{W}$ に含まれるキーワードを少なくとも 1 つ含むオブジェクトの集合 $O_{s.\mathcal{W}}$ の中で，o_q より s に近い位置に存在するオブジェクト，つまり，$s.l$ から距離 $Edist(s.l, o_q.l)$ 以内の範囲に含まれるオブジェクトの集合 O^+ を検索する．そして，$|O^+|$ が k を超えたとき，s は q の解になり得ないため，s のチェックを終了する．また，あるキーワード $w \in o_q.\mathcal{W}$ を含むサブスクリプション s_1 および s_2 の位置が近い場合，s_1 および s_2 は，同じオブジェクトが Top-k オブジェクトになりやすい．PART では，このようなサブスクリプションのチェックを同時に行うことにより，高速に解を計算する．

これを実現するため，サブスクリプション集合をキーワードごとに管理し，あるキーワード w を含むサブスクリプションの集合を互いに位置が近いもの同士で複数のグループに分割して管理する．このとき，w を含むサブスクリプションのグルー



△: サブスクリプション　　○: オブジェクト

(a) サブスクリプションのインデックス　(b) オブジェクトのインデックス

図 2: w_1 を含むサブスクリプション集合およびオブジェクト集合に対するインデックスの例

プの集合を $SG[w]$ とする．また，各サブスクリプションのグループ S_g は，そのグループに含まれるサブスクリプションの位置を包含する最小外接矩形 $S_g.mbr$ を保持する．また，あるART クエリ q が与えられ，キーワード $w \in o_q.\mathcal{W}$ を含むサブスクリプションのあるグループ S_g をチェックするとき，S_g に対して o_q より近い位置にあるオブジェクトの集合を高速に検索するため，オブジェクト集合をキーワードごとに R 木を用いて管理する．

例 2. 図 1 に示すようにサブスクリプションおよびオブジェクトが存在するとき，キーワード w_1 を含むサブスクリプションの集合およびオブジェクトの集合は，図 2 に示すように管理される．サブスクリプション集合は 4 つのグループに分割されており，$SG[w_1] = \{S_{g_1}, S_{g_2}, S_{g_3}, S_{g_4}\}$ である．

以下では，紹介したインデックスを用いたフィルタリング手法を紹介し，その後，PART の詳細について紹介する．

3.1 フィルタリング手法

PART では，ART クエリ $q = (o_q, k, \delta)$ が与えられたとき，キーワード $w \in o_q.\mathcal{W}$ を含む各サブスクリプションのグループ $S_g \in SG[w]$ において，S_g に含まれるすべてのサブスクリプションに対して，o_q より近い位置に存在するオブジェクトの集合 $S_g.O^+$ を検索する．つまり，$S_g.O^+$ に含まれるすべてのオブジェクト o は，$\forall s \in S_g, Edist(s.l, o.l) < Edist(s.l, o_q.l)$ を満たす．ここで，あるサブスクリプションまたはサブスクリプションのグループとあるオブジェクトまたは R 木のノード間の距離の上界値および下界値を $Edist_{ub}(\cdot,\cdot)$ および $Edist_{lb}(\cdot,\cdot)$ とする．このとき，あるサブスクリプションのグループ $S_g \in SG[w]$ および w に対する R 木のあるノード n に対して，次の定理が成り立つ．

定理 1. $Edist_{ub}(S_g.mbr, n) < Edist_{lb}(S_g.mbr, o_q.l)$ ならば，$\forall o \in O_n, o \in S_g.O^+$ である．ここで，O_n は，n を根とする部分木で管理されるオブジェクトの集合である．

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 1 より，$Edist_{ub}(S_g.mbr, n) < Edist_{lb}(S_g.mbr, o_q.l)$ を満たすノード n がいくつか存在し，$|S_g.O^+|$ が k を超えたとき，$\forall s \in S_g$ に対して，$o_q \notin s.T$ であるため，$q.ART$ に含まれなくてよい．そのため，$|S_g.O^+|$ が k 以上となった時点で，S_g のチェックを終了し，S_g に含まれるすべてのサブスクリプションをフィルタリングする．

また，次の定理も成り立つ．

定理 2. $Edist_{ub}(S_g.mbr, o_q.l) < \delta \cdot Edist_{lb}(S_g.mbr, n)$ ならば，$\forall o \in O_n$ は $\forall s \in S_g$ に対して $q.ART$ の正確性に影響を及ぼさない．

定理 2 より，$Edist_{ub}(S_g.mbr, o_q.l) < \delta \cdot Edist_{lb}(S_g.mbr, n)$ を満たすノード n を根とする部分木に含まれるすべてのオブジェクトをフィルタリングできる．

Algorithm 1: PART

Input: Q

1 **for** $\forall q \in Q$ **do**
2 　$q.ART \leftarrow \varnothing$
3 　**for** $\forall w \in o_q.\mathcal{W}$ **do**
4 　　**for** $\forall S_g \in SG[w]$ **do**
5 　　　$N_c, O^+ \leftarrow$ GetCandidates(q, w, S_g)
6 　　　**if** $|O^+| < q.k$ **then**
7 　　　　$q.ART \leftarrow$ $q.ART \cup$ Verification(q, w, S_g, N_c, O^+)
8 return each $q.ART$

3.2 アルゴリズム

アルゴリズム 1 は PART の概要を示している．ART クエリの集合 Q が与えられたとき，PART は，各クエリ $q \in Q$ の解を順に計算する．q の解を計算する際，キーワード $w \in o_q.\mathcal{W}$ を含むすべてのサブスクリプションのグループ $S_g \in SG[w]$ に対して以下の処理を行う．まず，GetCandidates により，w を含むオブジェクトの中で，o_q より S_g に近い位置に存在するオブジェクトの集合 O^+ および o_q より S_g に近い位置に存在するオブジェクトを含む可能性のあるノードの集合 N_c を検索する．その後，$|O^+|$ が k 未満である場合，Verification により，S_g に含まれるサブスクリプション s の中で，$q.ART$ に含まれるサブスクリプションを得る．

以下では，GetCandidates および Verification の詳細について述べる．

GetCandidates. アルゴリズム 2 は，キーワード w を含むあるサブスクリプションまたはサブスクリプションのグループ $\mathcal{S}$ に対して，w を含むオブジェクトの中で，o_q より近い位置に存在するオブジェクトの集合 O^+ および o_q より近い位置に存在するオブジェクトを含む可能性のあるノードの集合 N_c を検索するアルゴリズムを示している．w に対する R 木の根ノードから順に以下の操作を実行する．チェックを行うノード n が葉ノードである場合，n を N_c に加える（5–6 行）．一方，中継ノードである場合，n の子ノード n' に対して以下の処理を行う．$Edist_{ub}(\mathcal{S}.mbr, n) < Edist_{lb}(\mathcal{S}.mbr, o_q.l)$ である場合，定理 1 より，$\forall o \in O_{n'}$ は $\forall s \in \mathcal{S}$ に対して o_q より近い位置に存在するため，O^+ に追加する．さらに，$|O^+|$ が k 以上となった場合，$\mathcal{S}$ は，q の解に含まれなくてもよいためチェックを終了する

Algorithm 2: GetCandidates$(q, w, \mathcal{S})$

Input: $q = (o_q, k, c), w, \mathcal{S}$ // a subscription or a group of subscriptions

1 $O^+ \leftarrow \varnothing, N_c \leftarrow \varnothing, H \leftarrow \varnothing$
2 Push root node of R-tree for w into H
3 **while** $H \neq \varnothing$ **do**
4 $n \leftarrow$ the node popped from H
5 **if** *n is a leaf node* **then**
6 $N_c \leftarrow N_c \cup \{n\}$
7 **else**
8 **for** $\forall n' \in N$ *//N is n's children* **do**
9 **if** $Edist_{ub}(\mathcal{S}.mbr, n') < Edist_{lb}(\mathcal{S}.mbr, o_q.l)$ **then**
10 **for** $\forall o \in O_{n'}$ **do**
11 $O^+ \leftarrow O^+ \cup \{o\}$
12 **if** $|O^+| \geqq k$ **then**
13 return O^+, N_c
14 **else**
15 **if** $\delta \cdot Edist_{lb}(\mathcal{S}.mbr, n') \leqq Edist_{ub}(\mathcal{S}.mbr, o_q.l)$ **then**
16 Push n' into H
17 return O^+, N_c

(9–13 行)．ここで，$\mathcal{S}.mbr$ は，$\mathcal{S}$ がサブスクリプションのグループ S_g ならば $S_g.mbr$，サブスクリプション s ならば $s.l$ を表す．一方，$\delta \cdot Edist_{lb}(\mathcal{S}.mbr, n') > d_{ub}(\mathcal{S}.mbr, o_q.l)$ である場合，定理 2 より，n' を根とする部分木に含まれるすべてのオブジェクトは q の解に影響を及ぼさないためチェックを行う必要はない．そこで，$\delta \cdot Edist_{lb}(\mathcal{S}.mbr, n') \leqq Edist_{ub}(\mathcal{S}.mbr, o_q.l)$ を満たす場合のみ，再帰的に処理を行うため，n' を H に挿入する（14–16 行）．

Verification．アルゴリズム 3 は，S_g に含まれるサブスクリプション s の中で，ART に含まれるサブスクリプションを検索するアルゴリズムを示している．S_g に含まれる各サブスクリプション s に対して以下の処理を実行する．まず，S_g に対して検索された O^+ および N_c から，GetCandidates と同様の処理を行い，$s.O^+$ および $s.N_c$ を得る（3–10 行）．その後，w を除く s の各キーワード $w' \in s.\mathcal{W} \backslash \{w\}$ に対して，GetCandidates を実行し，$s.O^+$ および $s.N_c$ を得る．このとき，$|s.O^+|$ が k 以上となった場合，s は $q.ART$ に含まれなくてもよいため処理を終了する（11–14 行）．最後に，各キーワードに対する処理が終了した時点で，$|s.O^+|$ が k 未満である場合，s は ART に含まれる可能性があるため，CheckSubscription により，s が ART に含まれるかどうかチェックする（15–16 行）．ここで，CheckSubscription は，$s.O^+$ および $s.N_c$ に含まれるすべてのオブジェクトの中で s に対して o_q より近い位置に存在するオブジェクトの数が，k 未満であれば s を返し，k 以上であれば $NULL$ を返す関数である．

ここで，PART は $|s.O^+|$ が k 未満となるサブスクリプション s をクエリの解とするため，$o_q \in s.T$ である s は必ず解に含まれる．また，以下の定理が成り立つ．

Algorithm 3: Verification(q, w, S_g, N_c, O^+)

Input: q, w, S_g, N_c, O^+

1 $ART \leftarrow \varnothing$
2 **for** $\forall s \in S_g$ **do**
3 $s.O^+ \leftarrow O^+, s.N_c \leftarrow \varnothing$
4 **for** $\forall n \in N_c$ **do**
5 **if** $Edist_{ub}(s.l, n') < Edist_{lb}(s.l, o_q.l)$ **then**
6 **for** $\forall o \in O_{n'}$ **do**
7 $s.O^+ \leftarrow s.O^+ \cup \{o\}$
8 **else**
9 **if** $\delta \cdot Edist_{lb}(s.l, n') \leqq Edist_{ub}(s.l, o_q.l)$ **then**
10 $s.N_c \leftarrow s.N_c \cup \{n\}$
11 **for** $\forall w' \in s.\mathcal{W} \backslash w$ **do**
12 **if** $|s.O^+| \geqq k$ **then**
13 **break**
14 $s.O^+, s.N_c \leftarrow s.O^+, s.N_c \cup$ GetCandidates(q, w', s)
15 **if** $|s.O^+| < k$ **then**
16 $ART \leftarrow ART \cup$ CheckSubscription$(q, s, s.O^+, s.N_c)$
17 return ART

定理 3. PART は，δ が大きくなると高速化する．

4. B-PART

本章では，与えられた ART クエリの集合 Q に対して，各クエリの解を高速に求める新たなアルゴリズム B-PART（**B**atch **P**rocessing of **ART** クエリ）について紹介する．ここで，Q に含まれるクエリ q_1 および q_2 に対して，$o_{q_1}.\mathcal{W}$ および $o_{q_2}.\mathcal{W}$ が共通するキーワード w を含んでいると仮定する．このとき，PART は，Q に含まれる各 ART クエリの解を順に計算するため，w を含むサブスクリプションのグループ $S_g \in SG[w]$ に対して，複数回チェックをおこなってしまう．B-PART では，複数のクエリに共通するサブスクリプションのグループのチェックを同時に行うことにより，実行時間の削減を図る．これを実現するため，ART クエリの集合 Q が与えられたとき，Q に含まれるすべてのクエリオブジェクト o_q がもつキーワードの集合 $\mathcal{W}_Q$ を計算する．そして，各キーワード $w \in \mathcal{W}_Q$ に対する処理を順に行い，w を含む各サブスクリプションのグループを 1 度だけチェックする．

アルゴリズム 4 は B-PART の概要を示している．まず，与えられた ART クエリの集合 Q に対して，Q に含まれるすべてのクエリオブジェクト o_q がもつキーワードの集合 $\mathcal{W}_Q$ および各キーワード $w \in \mathcal{W}_q$ を含むクエリの集合 Q_w を計算する．そして，各キーワード $w \in \mathcal{W}_Q$ に対して順に処理を行う．具体的には，w を含むすべてのサブスクリプションのグループ $S_g \in SG[w]$ に対して以下の処理を行う．まず，BatchGetCandidates により，w を含むオブジェクトの中で，Q_w に含まれるすべてのクエリオブジェクトより S_g に近い位置に存在するオブジェクトの集合 O^+ および S_g に近い位置に存在するオブジェ

Algorithm 4: B-PART

```
Input: Q
1  W_Q ← ∅, Q_· ← ∅
2  for ∀q ∈ Q do
3      q.ART ← ∅
4      for ∀w ∈ o_q.W do
5          W_Q ← w, Q_w ← q
6  for ∀w ∈ W_Q do
7      for ∀S_g ∈ SG[w] do
8          N_c, O+ ← BatchGetCandidates(Q_w, w, S_g)
9          if |Q_w| > 0 ∧ |O+| < k_max
           //k_max = max{q.k|q ∈ Q_w} then
10             Q_w.ART ← Q_w.ART ∪
               BatchVerification(Q_w, w, S_g, N_c, O+)
11 return each q.ART
```

クトを含む可能性のあるノードの集合 N_c を検索する．その後，$|O^+|$ が k_{max} 未満である場合，BatchVerification により，S_g に含まれるサブスクリプション s の中で，Q_w に含まれる各クエリ q の解 $q.ART$ に含まれるサブスクリプションを得る．ここで，k_{max} は，Q_w に含まれる各クエリ q の $q.k$ の中で最大の値であり，$Q_w.ART$ は，$q.ART$（$q \in Q_w$）の集合を表す．

以下では，BatchGetCandidates および BatchVerification の詳細について述べる．

BatchGetCandidates．アルゴリズム 5 は，w を含むオブジェクトの中で，Q_w に含まれるすべてのクエリオブジェクトに対して，$\mathcal{S}$ に近い位置に存在するオブジェクトの集合 O^+ および $\mathcal{S}$ に近い位置に存在するオブジェクトを含む可能性のあるノードの集合 N_c を検索するアルゴリズムを示している．GetCandidates と同様に，w に対する R 木の根ノードから順に操作を実行する．ある子ノード n' に対する処理では，まず，$Edist_{ub}(\mathcal{S}.mbr, n')$ と $Edist_{min}$ を比較する（12 行）．ここで，$Edist_{min}$ は，Q_w に含まれる各クエリオブジェクトに対する $Edist_{lb}(\mathcal{S}.mbr, o_q.l)$ の最小値であるため，$Edist_{ub}(\mathcal{S}.mbr, n') < Edist_{min}$ ならば，$\forall q \in Q_w, Edist_{ub}(\mathcal{S}.mbr, n') < Edist_{lb}(\mathcal{S}.mbr, o_q.l)$ が成り立つ．よって，$O_{n'}$ に含まれるすべてのオブジェクトを O^+ に追加する．さらに，$k_{max} = \max\{q.k | q \in Q_w\}$ であるため，$|O^+|$ が k_{max} 以上ならば，$\mathcal{S}$ は，$\forall q \in Q_w$ の解に含まれなくてもよいためチェックを終了する（15–16 行）．

一方，$Edist_{ub}(\mathcal{S}.mbr, n') < Edist_{min}$ でない場合，以下の処理を行う．Q_w は $Edist_{ub}(\mathcal{S}.mbr, o_q.l)$ の降順にソートされているため，先頭のクエリ q' ほど，$\mathcal{S}$ に対して，$o_{q'}$ より近い位置に存在するオブジェクトが多く存在する．そこで，$|O^+ \cup O_{n'}| \geqq q'.k$ である場合，$Edist_{ub}(\mathcal{S}.mbr, n')$ と $Edist_{lb}(\mathcal{S}.mbr, o_{q'}.l)$ を比較する．$Edist_{ub}(\mathcal{S}.mbr, n') < Edist_{lb}(\mathcal{S}.mbr, o_{q'}.l)$ ならば，$o_{q'}$ より近い位置に存在するオブジェクトが $q'.k$ 個以上存在するため，$\mathcal{S}$ は q' の解に含まれなくてもよい．そのため，q' を Q_w から取り除き，次のクエリに対して同様の処理を行う（18–22 行）．一方，$|O^+ \cup O_{n'}| \geqq q'.k$ でない場合，$\delta_{min} \cdot Edist_{lb}(\mathcal{S}.mbr, n') \leqq Edist_{ub}(\mathcal{S}.mbr, o_{q'}.l)$ ならば，n'

Algorithm 5: BatchGetCandidates$(Q_w, w, \mathcal{S})$

```
Input: Q_w, w, S
1  Sort Q_w in descending order of Edist_ub(S.mbr, o_q.l)
2  O+ ← ∅, N_c ← ∅, H ← ∅, k_max ← max{q.k|q ∈
   Q_w}, δ_min ← min{q.δ|q ∈ Q_w}
3  Edist_min ← min{Edist_lb(S.mbr, o_q.l)|q ∈ Q_w}
4  Push root node of R-tree for w into H
5  q' ← the first query of Q_w
6  while H ≠ ∅ do
7      n ← the node popped from H
8      if n is a leaf node then
9          N_c ← N_c ∪ {n}
10     else
11         for ∀n' ∈ N //N is n's children do
12             if Edist_ub(S.mbr, n') < Edist_min then
13                 for ∀o ∈ O_n' do
14                     O+ ← O+ ∪ {o}
15                 if |O+| ≧ k_max then
16                     return O+, N_c
17             else
18                 while
                   |O+ ∪ O_n'| ≧ q'.k ∧ Edist_ub(S.mbr, n') <
                   Edist_lb(S.mbr, o_q'.l) do
19                     Q_w ← Q_w\{q'}
20                     if Q_w = ∅ then
21                         return O+, N_c
22                     q' ← the first query of Q_w
23                 if δ_min · Edist_lb(S.mbr, n') ≦
                   Edist_ub(S.mbr, o_q'.l) then
24                     Push n' into H
25 return O+, N_c
```

を H に挿入する（23–24 行）．ここで，$\delta_{min} = \min\{q.\delta | q \in Q_w\}$ である．

BatchVerification．アルゴリズム 6 は，S_g に含まれるサブスクリプション s の中で，$Q_w.ART$ に含まれるサブスクリプションを検索するアルゴリズムを示している．Verification と同様に，S_g に含まれる各サブスクリプション s に対して処理を実行する．$s.O^+$ および $s.N_c$ を得る際，$Edist_{min}$，$Edist_{max}$，および δ_{min} に基づいて処理を行う（7–13 行）．w を除く s の各キーワード $w' \in s.\mathcal{W}\backslash\{w\}$ に対する処理が終了した後，$|s.O^+|$ が k_{max} 未満である場合，Q_w に含まれる各クエリに対して，CheckSubscription を実行する（18–20 行）．

5. PB-PART

本章では，与えられた ART クエリの集合に対する処理をより高速に行うため，B-PART をマルチコア環境で行うアルゴリズム PB-PART（**P**arallel **B**atch **P**rocessing of **ART** クエリ）を紹介し，負荷分散を行う方法について議論する．アルゴリズム 4 の 6 行目以降，B-PART は，各キーワード $w \in \mathcal{W}_Q$ に対して順に処理を行う．これらの処理は互いに独立している

Algorithm 6: BatchVerification(Q_w, w, S_g, N_c, O^+)

Input: Q_w, w, S_g, N_c, O^+

$Q_w.ART \leftarrow \varnothing$
$k_{max} \leftarrow \max\{q.k|q \in Q_w\}, \delta_{min} \leftarrow \min\{q.\delta|q \in Q_w\}$
for $\forall s \in S_g$ **do**
  $s.O^+ \leftarrow O^+, s.N_c \leftarrow \varnothing$
  $Edist_{min} \leftarrow \min\{Edist(s.l, o_q.l)|q \in Q_w\}$
  $Edist_{max} \leftarrow \max\{Edist(s.l, o_q.l)|q \in Q_w\}$
  for $\forall n \in N_c$ **do**
    if $Edist_{ub}(s.l, n') < Edist_{min}$ **then**
      for $\forall o \in O_{n'}$ **do**
        $s.O^+ \leftarrow s.O^+ \cup \{o\}$
    else
      if $\delta_{min} \cdot Edist_{lb}(s.l, n') \leqq Edist_{max}$ **then**
        $s.N_c \leftarrow s.N_c \cup \{n\}$
  for $\forall w' \in s.\mathcal{W} \backslash w$ **do**
    if $|s.O^+| \geqq k_{max}$ **then**
      break
    $s.O^+, s.N_c \leftarrow$
     $s.O^+, s.N_c \cup$ BatchGetCandidates(Q_w, w', s)
  if $|s.O^+| < k_{max}$ **then**
    for $\forall q \in Q_w$ **do**
      $q.ART \leftarrow$
       $q.ART \cup$ CheckSubscription$(q, s, s.O^+, s.N_c)$
return $Q_w.ART$

ため，並列して行うことを考える．各キーワード w に対する処理では，w を含むサブスクリプションのグループに対して，クエリオブジェクトより近い位置に存在するオブジェクトを検索する．そのため，各キーワードに対する処理時間はそのキーワードを含むオブジェクトの分布に依存する．w が多くのオブジェクトに含まれる場合，検索するオブジェクトの数が増加するため，処理時間は大きくなる．さらに，提案手法における R 木では，オブジェクトが多く存在する領域を再帰的に分割するため，オブジェクトが密集している領域の木の深さが深くなる．そのため，その領域を検索するとき，処理時間が増加する．

以上の特徴に基づき，各キーワード w に対する処理時間を予測するため，w に対する処理のコスト $cost(w)$ を以下のように定義する．

$$cost(w) = \sum_{\forall n \in LN_w} \phi^{n.depth} \cdot |O_n| \qquad (1)$$

ここで，LN_w は w に対する R 木に含まれる葉ノードの集合，$n.depth$ はノード n の深さ，および ϕ はノードの深さに対する重みである．

式（1）により計算されたコストに基づき，各キーワードを処理するコアを決定する方法について紹介する．まず，以下の定理が成り立つ．

定理 4. $\mathcal{W}_Q$ が与えられたとき，各コア間の処理時間の差を最小化するキーワードの分散を決定することは，NP 困難である．

表 1: データセットの詳細

データセット	*TWEETS*	*PLACES*
オブジェクトの数	20M	9.4M
キーワードの種類	2.1M	54K
あるオブジェクトに含まれる平均のキーワードの数	5.2	2.9

最適なキーワードの分散を決定することは NP 困難であるため，PB-PART では，$\mathcal{W}_Q$ が与えられたとき，貪欲法を用いて分散を決定する．

6. 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，18 core Intel Xeon Gold 6154 (3.0GHz)，および 512GB 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．また，マルチコア処理を行うため，OpenMP を用いた．

6.1 セッティング

本実験では，シングルコア環境での性能を評価するため，以下のアルゴリズムの性能を評価した．

- **B-PART**. 4 章で紹介したアルゴリズム．
- **PART**. 3 章で紹介したアルゴリズム．
- **RG**. 文献 [12] で提案された手法を本問題に適用したもの．この手法を本問題に適用した場合，オブジェクト集合を 1 つの IR 木で管理する．そして，ART クエリ $q = \{o_q, k, \delta\}$ が与えられたとき，o_q が Top-k オブジェクトになり得るすべてのサブスクリプションに対して，Top-k 検索を実行する．Top-k 検索の途中で，o_q を Top-k オブジェクトに含む可能性が無いと判断された時点で検索を終了する．

また，マルチコア環境での性能を評価するため，以下のアルゴリズムの性能を評価した．

- **PB-PART**. 5 章で紹介したアルゴリズム．
- **PB-PART-naive**. 5 章で紹介したアルゴリズムにおいて，計算された W_Q に含まれる各キーワードをハッシュパーティションして処理を行う手法．
- **P-PART-naive**. 3 章で紹介したアルゴリズムにおいて，与えられた Q に含まれるクエリをハッシュパーティションして処理を行う手法．

データセット. 本実験では，オブジェクトとして 2 つの実データ（*TWEETS* [1] および *PLACES* [7]）を用いた．表 1 に，データセットの詳細を示す．

サブスクリプション. オブジェクトのデータセットに基づいて，$|S|$ 個のサブスクリプションを作成した．各サブスクリプションは，1 つのツイートまたはエントリの中に現れるキーワードの中から l 個のキーワードをランダムに選び，サブスクリプションのキーワードとする（$1 \leqq l \leqq 5$）．また，サブスクリプションの位置は選ばれたツイートまたはエントリの位置とした．

クエリ. ある ART クエリ q は，オブジェクト集合に含まれるオブジェクトをランダムに選びクエリオブジェクト o_q とする．また，$q.k$ は 1 から k_{max} の間の一様乱数とし，$s.\delta$ は $\overline{\delta}$ を平均

表 2: パラメータの設定

パラメータ	値
サブスクリプションの数，$\|S\|$ [$\times 10^6$]	**0.25**, 0.5, 0.75, 1
$q.k$ の最大値，k_{max}	5, **10**, 15, 20, 25, 30
近似率，$\overline{\delta}$	1, **1.5**, 2, 2.5, 3, 3.5, 4
一度に与えられるクエリの数，$\|Q\|$ [$\times 10^3$]	**1**, 2.5, 5, 7.5, 10
スレッドの数，$\|\mathcal{T}\|$	**1**, 3, 6, 9, 12, 15, 18

表 3: 各手法における平均計算時間 [sec]

手法	*TWEETS*	*PLACES*
RG	30.168	38.185
PART	1.453	1.748
B-PART	0.777	0.856

とする正規分布に従うとする．

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，オブジェクトの数 $|O| = 1,000,000$ とした．

6.2 評価結果

本実験では，ART クエリの集合が 100 回与えられたときの各アルゴリズムにおける平均計算時間（与えられた ART クエリの集合に対して，各クエリの解の計算が完了するまでの時間の平均値）の結果を紹介する．

6.2.1 シングルコア処理

各手法の性能の比較．表 3 に，各アルゴリズムにおける平均計算時間の結果を示す．使用したデータセットによらず，提案アルゴリズム（PART および B-PART）は高速に解を計算できることがわかる．また，各サブスクリプションのグループに対する重複するチェックを削減するため，B-PART は PART と比べて高速に解を計算できる．ここで，*TWEETS* を使用した場合よりも *PLACES* を使用した場合の方が平均計算時間が大きくなる．これは，*TWEETS* に比べて *PLACES* の方がキーワードの種類が少なく，各オブジェクトおよびサブスクリプションが同じキーワードを含みやすくなる．その結果，与えられたクエリに対して，チェックを行うサブスクリプションの数および各サブスクリプションのチェックにおいて評価するオブジェクトの数が多くなるため，平均計算時間が大きくなる．各パラメータの影響を評価した実験では，*TWEETS* を使用して比較を行った．

k_{max} の影響．図 3 にクエリの指定する k の最大値 k_{max} を変えたときの結果を示す．提案アルゴリズムは，RG と比較して，常に高速に解を計算できることがわかる．また，k_{max} の増加に伴い，各アルゴリズムにおいて平均計算時間が増加するが，提案アルゴリズムの平均計算時間の増加率の方が小さく，k_{max} が大きい場合，提案アルゴリズムはより有用であることがわかる．

$|S|$ の影響．図 4 にサブスクリプションの数 $|S|$ を変えたときの結果を示す．提案アルゴリズムは，RG と比較して，常に高速に解を計算できることがわかる．また，$|S|$ の増加に伴い，各アルゴリズムにおいて平均計算時間が増加し，さらに，提案アルゴリズムの平均計算時間の増加率は RG と比較して小さいことがわかる．RG は評価を行う必要のあるサブスクリプション

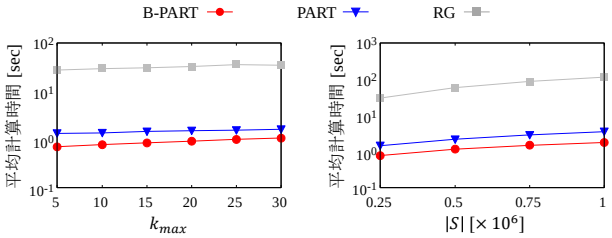


図 3: k_{max} の影響

図 4: $|S|$ の影響

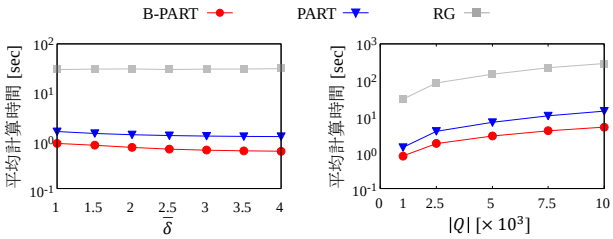


図 5: $\overline{\delta}$ の影響

図 6: $|Q|$ の影響

を順にチェックするのに対し，提案アルゴリズムは，サブスクリプションをグループに分割し，グループ単位でチェックを行う．その結果，チェックを行う回数を削減できるため，$|S|$ の増加による影響が小さくなる．

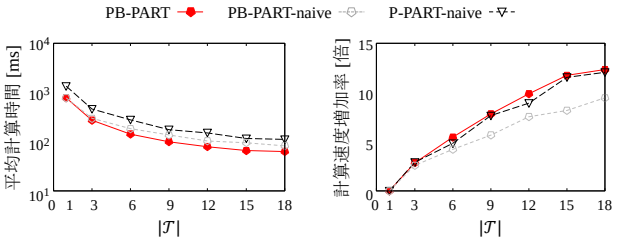
$\overline{\delta}$ の影響．図 5 に近似率 $\overline{\delta}$ を変えたときの結果を示す．提案アルゴリズムは，RG と比較して，常に高速に解を計算できることがわかる．また，$\overline{\delta}$ が増加すると，RG の平均計算時間はほぼ一定であるのに対し，提案アルゴリズムの平均計算時間は減少することがわかる．これは，$\overline{\delta}$ が増加すると，定理 2 の条件を満たすノードの数が増加し，より多くのノード（そのノードを根とする部分木に含まれるオブジェクト）をチェックする必要がなくなるためである．

$|Q|$ の影響．図 6 に一度に与えられるクエリの数 $|Q|$ を変えたときの結果を示す．提案アルゴリズムは，RG と比較して，常に高速に解を計算できることがわかる．また，$|Q|$ の増加に伴い，各アルゴリズムにおいて平均計算時間が増加することがわかる．さらに，$|Q|$ が増加すると，B-PART と PART の平均計算時間の差が大きくなることがわかる．$|Q|$ が増加すると，与えられたクエリ集合の中で，共通するキーワードをもつクエリオブジェクトの数が増加する．そのため，B-PART は PART と比較して，各サブスクリプションのグループに対するチェックの回数をより多く削減できるため，平均計算時間の差が大きくなる．

6.2.2 マルチコア処理

図 7a に，使用する CPU コアの数 $|\mathcal{T}|$ を変えたときの結果を示す．PB-PART および PB-PART-naive を比較すると，PB-PART の方が $|\mathcal{T}|$ の増加に対して，平均計算時間が減少することがわかり，提案したコストモデルに基づくキーワードの分散が有効であることを示している．また，PB-PART および P-PART-naive を比較すると，PB-PART は常に高速に解を計算できることがわかる．

さらに，図 7b に，各アルゴリズムのシングルコア処理に対する計算速度の増加率を示す．PB-PART-naive および P-



(a) 計算時間　　(b) 計算速度

図 7: $|\mathcal{T}|$ の影響

PART-naive を比較すると，PB-PART-naive の方が増加率が小さいことがわかる．P-PART-naive は，与えられたクエリ集合に含まれるクエリをハッシュパーティションして処理を行う．このとき，各クエリに対する処理時間が大きく異なることは少ないため，各コアにおける処理時間が大きく異なることは少ない．一方，PB-PART-naive は，計算されたキーワード集合に含まれるキーワードをハッシュパーティションして処理を行う．このとき，各キーワードに対する処理時間はそのキーワードを含むオブジェクトおよびサブスクリプションの数に依存して大きく異なる．そのため，処理時間の大きいキーワードを多く処理するコアにおける処理時間が大きくなる．その結果，PB-PART-naive の計算時間の増加率は小さくなる．PB-PART は，PB-PART-naive における問題を解決するため，コストモデルに基づいてキーワードを分散する．これにより，各コアにおける処理時間の差を小さくすることができるため，P-PART-naive と同程度の計算速度の増加率を達成できる．

7. 関 連 研 究

本章では，逆 Top-k 検索に関する従来研究，および近似逆最近傍検索に関する従来研究を紹介する．

逆 Top-k 検索． これまでに，多くの研究が逆 Top-k 検索に関する問題に取り組んでいる．文献 [10], [11] では，位置情報に基づき逆 k 最近傍検索を行う問題に取り組んでいる．しかし，これらの文献はキーワードを考慮していないため，本章における問題とは異なる．一方，様々な研究が位置およびキーワードに基づく逆 Top-k 検索に関する問題に取り組んでいる [3], [6], [12]．例えば，文献 [12] では，位置情報およびキーワードに加えてソーシャル関係に基づく逆 Top-k 検索の解を検索する問題に取り組んでいる．具体的には，ユーザ集合およびオブジェクト集合を GIM-tree と呼ばれる木構造で管理し，クエリの解となり得るユーザの集合を限定することで，効率的に解を計算する．しかし，この研究では，距離関数として道路ネットワーク上での距離を想定しているため，本章における問題とは異なる．

近似逆最近傍検索． 文献 [4], [5] において，近似逆最近傍検索に関する問題に取り組んでいる．これらの文献では，各サブスクリプション s に対して最近傍であるオブジェクトまでの距離を $Edist_{nn}(s)$ をしたとき，あるクエリオブジェクト o_q に対して $Edist(s.l, o_q.l) \leq \delta \cdot Edist_{nn}(s)$（$\delta > 1$）を満たすすべてのサブスクリプションを検索する．しかし，これらの文献は，Top-k 検索を対象としていない．さらに，キーワードも考慮していないため，本章における問題とは異なる．

8. ま と め

本稿では，位置およびキーワードに基づく近似逆 Top-k クエリ（ART クエリ）処理問題に取り組み，ART クエリの集合が与えられたとき，各クエリの解を効率的に検索するアルゴリズム（PART）を紹介した．また，PART を拡張し，与えられたクエリ集合に対してバッチ処理を行うアルゴリズム（B-PART）を提案した．さらに，マルチコア環境で処理を行うアルゴリズム（PB-PART) も提案した．実データを用いた実験の結果から，提案アルゴリズムは，高速に解を検索できることを確認した．

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)(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, Z., Cong, G., Zhang, Z., Fuz, T.Z., Chen, L.: Distributed publish/subscribe query processing on the spatio-textual data stream. In: ICDE. pp. 1095–1106 (2017)
[3] Choudhury, F.M., Culpepper, J.S., Sellis, T., Cao, X.: Maximizing bichromatic reverse spatial and textual k nearest neighbor queries. PVLDB 9(6), 456–467 (2016)
[4] Hidayat, A., Cheema, M.A., Taniar, D.: Relaxed reverse nearest neighbors queries. In: SSTD. pp. 61–79 (2015)
[5] Hidayat, A., Yang, S., Cheema, M.A., Taniar, D.: Reverse approximate nearest neighbor queries. TKDE 30(2), 339–352 (2018)
[6] Lu, J., Lu, Y., Cong, G.: Reverse spatial and textual k nearest neighbor search. In: SIGMOD. pp. 349–360 (2011)
[7] Mahmood, A.R., Aly, A.M., Aref, W.G.: Fast: Frequency-aware indexing for spatio-textual data streams. In: ICDE. pp. 305–316 (2018)
[8] Wang, X., Zhang, W., Zhang, Y., Lin, X., Huang, Z.: Top-k spatial-keyword publish/subscribe over sliding window. VLDBJ 26(3), 301–326 (2017)
[9] Wang, X., Zhang, Y., Zhang, W., Lin, X., Wang, W.: Ap-tree: efficiently support location-aware publish/subscribe. VLDBJ 24(6), 823–848 (2015)
[10] Yang, S., Cheema, M.A., Lin, X., Wang, W.: Reverse k nearest neighbors query processing: experiments and analysis. PVLDB 8(5), 605–616 (2015)
[11] Yang, S., Cheema, M.A., Lin, X., Zhang, Y.: Slice: reviving regions-based pruning for reverse k nearest neighbors queries. In: ICDE. pp. 760–771 (2014)
[12] Zhao, J., Gao, Y., Chen, G., Jensen, C.S., Chen, R., Cai, D.: Reverse top-k geo-social keyword queries in road networks. In: ICDE. pp. 387–398 (2017)