Approximate Reverse Top-k Spatial-Keyword Queries

Shunya Nishio
Osaka University
Osaka, Japan
shuntdg@gmail.com

Daichi Amagata
Osaka University
Osaka, Japan
amagata.daichi@ist.osaka-u.ac.jp

Takahiro Hara
Osaka University
Osaka, Japan
hara@ist.osaka-u.ac.jp

***Abstract*—Location-based services are becoming more involved with our daily lives, so many works have considered efficiently retrieving useful objects from spatial-keyword databases. These works are promising on the user sides, but none of them considers the service provider sides. To gain profits and enrich recommendation lists, service providers conduct market analyses and want to know potential users who may be interested in their services. In this paper, to satisfy this requirement, we propose a new query, approximate reverse top-k spatial-keyword (ART) query. Given a set O of spatial-keyword objects, a set S of users (their locations and preferable keywords), a query object q, k, and an approximation ratio ϵ, an ART query retrieves such users that q is included in their approximate top-k results among O and q. A straightforward approach to processing this query is to run a top-k spatial-keyword search for each user in S. This is clearly expensive, as the number of users is generally large. We therefore propose PART, an efficient algorithm for ART query processing. In addition, we propose B-PART, which enables the processing of multiple ART queries in a batch. We conduct extensive experiments using real datasets, and the results demonstrate the efficiencies of our algorithms.**

***Index Terms*—spatial-keyword data, reverse top-k query, batch processing**

I. Introduction

Motivation and Challenge. Recently, many objects that contain spatial information have been generated because of the proliferation of GPS-enabled devices [1]–[8]. In addition, they can contain keywords, as represented by social networking service applications [9]–[15]. These *spatial-keyword* objects are used in location-related applications, such as POI search [16] and personalized advertisements [17]. In these applications, a set O of spatial-keyword objects, which are obtained by service providers, is managed by a system, and users issue top-k spatial-keyword queries in this system [16], [18], [19] to retrieve their required objects from O. Users hence receive benefits from this system, but the service providers may have a question: how to find users who may be interested in their services? Analyzing the statistical characteristics of potential customers actually helps market analysis, developing new products, getting new customers, and enabling up-selling and cross-selling [20], [21]. Nevertheless, few works have considered how to find such potential customers (users) from spatial-keyword database management systems.

Motivated by this, this paper proposes a new query, namely *approximate reverse top-k spatial-keyword query* (ART query in short). Given a set O of spatial-keyword objects and a set S of users (their locations and preferable keywords), let $dist(o, s)$ be the distance between $o \in O$ and $s \in S$. Then, given O, S, and a query object q (i.e., location, keywords, k, and an approximation ratio $\epsilon \geq 1$), an ART query retrieves users $s \in S$, where they have at least one common keyword with q and $dist(q, s) \leq \epsilon \cdot dist(o^*, s)$. Note that o^* is the k-th nearest neighbor of s among a set of spatial-keyword objects having at least a keyword in common with s. (The formal definition is presented in Section II-A.) For example, assume that q represents a restaurant, and this query finds users who are interested in this restaurant. ART queries employ an approximation parameter ϵ, because q does not necessarily have to be in the top-k list for s in practice and we can avoid tie-breaks in the ranking. For example, if $dist(q, s) > dist(o^*, s)$ but $dist(q, s) \approx dist(o^*, s)$, the importance (or availability) of q for the user s would be nearly the same as that of the top-k spatial-keyword objects [22]. ART queries *can* include such users in the result sets (see Definition 3 for detail).

Contribution. We therefore propose an efficient ART query processing algorithm, PART (algorithm for Processing an ART query), which incorporates computation sharing and filtering techniques. Moreover, we extend PART to enable the processing of multiple ART queries in a batch, and propose B-PART (Batch-PART). It is usual that multiple queries are issued at the same time [17], [19]. For example, a service provider issues multiple ART queries of different locations (services), and some restaurants may issue ART queries at noon to investigate their potential diners. Running PART for each ART query incurs a delay for not-yet-processed ART queries, and B-PART avoids this drawback.

To summarize, our contributions are as follows:

- We propose a new query, namely approximate reverse top-k spatial-keyword (ART) query. To the best of our knowledge, this is the first work that considers ART queries.
- We provide a baseline algorithm for ART query processing.
- We propose PART, a more sophisticated algorithm than the baseline one.
- We propose B-PART, which efficiently processes multiple ART queries in a batch.
- We conduct experiments using real datasets, and the results

demonstrate that (i) PART is faster than the baseline algorithm and (ii) B-PART accelerates the processing of ART queries when they are issued at the same time.

Road-map. Section II formally defines our problem and introduces a baseline algorithm. In Section III, we propose our algorithm, PART. We extend PART and propose B-PART to enable batch processing of ART queries in Section IV. We report our experimental results in Section V. Related works are reviewed in Section VI. Finally, in Section VII, we conclude this paper.

II. Preliminary

This section first defines our problem, and then designs a baseline algorithm. Table I describes the notations frequently used in this paper.

A. Problem Definition

We use $o = \langle l, W \rangle$ to denote a spatial-keyword object, where l and W are a two-dimensional location information (latitude and longitude) on o and a set of keywords held by o, respectively. Similarly, we use $s = \langle l, W \rangle$ to denote the information on a user s, where l and W are a two-dimensional location information on s and a set of keywords held by s, respectively. As with existing works [9], [11], [14], [19], [23], the locations and keywords held by spatial-keyword objects and users are static. Let $dist(o, s)$ be the Euclidean distance between $o.l$ and $s.l$. In addition, let O and S be respectively sets of spatial-keyword objects and users. We assume that O and S are memory-resident [9], [11], [14], [20], [23]–[26].

For ease of explanation, we first define the top-k spatial-keyword objects for a given user [27], [28].

Definition 1 (Top-k spatial-keyword objects). *Given O, a user s, and a result size k, the set T_s of the top-k spatial-keyword objects for s satisfies the following: (i) $T_s \subseteq O_{s.W} = \{o \,|\, o \in O, o.W \cap s.W \neq \varnothing\}$, (ii) $|T_s| = k$[1], and (iii) $\forall o \in T_s, \forall o' \in O_{s.W} \backslash T_s, dist(o, s) \leq dist(o', s)$ (ties are broken arbitrarily).*

From the above definition, it can be seen that the top-k spatial-keyword objects for s are the k nearest neighbor objects among the set of spatial-keyword objects having at least one keyword in common with s.

Next, based on Definition 1, we define *reverse top-k spatial-keyword queries*.

Definition 2 (Reverse top-k spatial-keyword query). *Given O, S, and $q = \langle l, W, k \rangle$, a reverse top-$k$ spatial-keyword query (RT query in short) retrieves a set of users $s \in S$ such that $\langle q.l, q.W \rangle \in T_s$ among $O \cup \{\langle q.l, q.W \rangle\}$.*

Note that $\langle q.l, q.W \rangle$ can be in O. Without loss of generality, we assume $\langle q.l, q.W \rangle \in O$, so $O \cup \{\langle q.l, q.W \rangle\} = O$. Let RT_q denote the answer set of the RT query of q, i.e., $RT_q = \{s \,|\, s \in S, \langle q.l, q.W \rangle \in T_s\}$. We below relax Definition 2, based on the rationale introduced in Section I.

[1]We assume $|O_{s.W}| \geq k$. Otherwise, $T_s = O_{s.W}$.

TABLE I
Notations frequently used in this paper

Notation	Meaning
O	A set of spatial-keyword objects
o	A spatial-keyword object
S	A set of users
s	A user (a pair of location and keyword set)
$dist(o, s)$	The Euclidean distance between o and s
q	An ART query (location, keyword set, k, and ϵ)
l	Location information (on o s, or q)
W	A set of keywords
w	A keyword
ϵ	An approximation ratio
n	A node of an R-tree

Definition 3 (Approximate reverse top-k spatial-keyword query). *Given O, S, and $q = \langle l, W, k, \epsilon \rangle$ ($\epsilon \geq 1$), an approximate reverse top-k spatial-keyword query (ART query in short) identifies its answer, denoted by ART_q, based on the following criteria:*

1) *If $s \in RT_q$, $s \in ART_q$.*
2) *If $s \notin RT_q$ but $dist(q, s) \leq \epsilon \cdot dist(o^*, s)$ where o^* is the top k-th spatial-keyword object for s, s "can" be in ART_q.*
3) *If $s \notin RT_q$ and $\epsilon \cdot dist(o^*, s) < dist(q, s)$, $s \notin ART_q$.*

From the above definition, it is important to note the following. Condition (1) guarantees that the result of an ART query certainly contains the result of the RT query with the same parameters. Condition (2) allows users s such that $s \notin RT_q$ but $dist(q, s) \leq \epsilon \cdot dist(o^*, s)$ to be included in the result of an ART query[2]. Condition (3) defines the users who cannot be in ART_q. Last, RT queries are a special version of ART queries with $\epsilon = 1$.

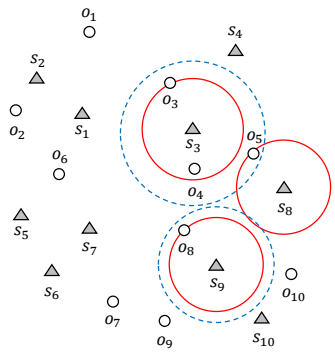
Example 1. *Fig. 1 illustrates 10 users and 10 spatial-keyword objects, given an ART query q, where $\langle q.l, q.W \rangle = o_5$ and $k = 1$. Because o_5 is the top-1 spatial-keyword object for s_8, $s_8 \in ART_q$. Similarly, the top-1 spatial-keyword object for s_3 (s_9) is o_3 (o_8). In this figure, the blue dotted circles are extended from their corresponding red circle by a factor of $q.\epsilon$. Notice that we have $dist(o_5, s_3) \leq \epsilon \cdot dist(o_3, s_3)$ (the blue dotted circle centered at s_3), so s_3 can be in ART_q. On the other hand, we have $\epsilon \cdot dist(o_8, s_9) \leq dist(o_5, s_9)$ (the blue dotted circle centered at s_9), so s_9 cannot be in ART_q.*

In this paper, we solve the following problems exactly while minimizing computation time in Sections III and IV, respectively.

Problem 1 (ART query processing). *Given O, S, and an ART query q, we compute ART_q so that it follows Definition 3.*

Problem 2 (Batch ART query processing). *Given O, S, and a set Q of ART queries, we compute ART_q of each $q \in Q$ so that it follows Definition 3.*

[2]Notice that this is a "don't care" case, i.e., users having condition (2) may or may not be included in ART_q.



s	Keywords
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	Keywords
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

Fig. 1. Example of a set of users and a set of spatial-keyword objects. Spatial-keyword objects and users are respectively represented by ○ and △. Left part shows the coordinates of spatial-keyword objects and users, whereas right part shows the keywords held by each spatial-keyword object and user.

B. Baseline Algorithm

As this is the first work that addresses the ART query processing problem, we first design a baseline algorithm (for Problem 1). To start with, we focus on an RT query q. An important observation supporting efficient (A)RT query processing is that, for a given s, we do not need to retrieve its exact top-k spatial-keyword objects but *it is sufficient to know whether* $q \in T_s$ *or not*. This is because we can know $s \notin RT_q$ whenever we know $q \notin T_s$. This *decision version* of top-k spatial-keyword search is formally defined as follows:

PROBLEM 3 (TOP-k SPATIAL-KEYWORD DECISION PROBLEM). *Given O, where $\langle q.l, q.W\rangle \in O$, a user s, and a result size k, this problem returns yes (resp. no) iff $\langle q.l, q.W\rangle \in T_s$ (resp. $\langle q.l, q.W\rangle \notin T_s$).*

This problem can be solved by finding k (or more) spatial-keyword objects in $O_{s.W}$ with shorter distances than $dist(q, s)$ (the distance between $q.l$ and $s.l$). To achieve this, we find a set $O^+_{s.W} = \{o \,|\, o \in O_{s.W}, dist(o, s) < dist(q, s)\}$. If $|O^+_{s.W}| \geq k$, we can ignore s. To efficiently find $O^+_{s.W}$, our baseline algorithm builds a tree index (e.g., kd-tree [29] or R-tree [30]) for each set of objects having a keyword w, similar to [19].

Algorithm description. Algorithm 1 describes our baseline algorithm. In a nutshell, given a user s such that $s.W \cap q.W \neq \varnothing$, it incrementally updates $O^+_{s.W}$. Lines 3–10 correspond to solving the top-k spatial-keyword decision problem for s. By exploiting the tree structure, for each keyword $w \in s.W$, this algorithm incrementally obtains nearest neighbor objects o such that $w \in o.W$ and $dist(o, s) < dist(q, s)$, and adds them into $O^+_{s.W}$. When $|O^+_{s.W}| \geq k$, it terminates evaluating s. If s still has $|O^+_{s.W}| < k$ after evaluating s w.r.t. all keywords in $s.W$, s is added into RT_q. This is repeated for each user.

Extension for ART queries. We use the following theorem to apply this baseline algorithm for ART query processing.

THEOREM 1. *Given an ART query q and a user s such that $s.W \cap q.W \neq \varnothing$, s has two cases. Case (i) there are at least k spatial-keyword objects o having $\epsilon \cdot dist(o, s) < dist(q, s)$: s cannot be in ART_q. Case (ii) otherwise: s has either condition (1) or (2) of Definition 3.*

Algorithm 1: Baseline

Input: O, S, and q
Output: RT_q

```
1  RT_q ← ∅
2  for each s ∈ S such that s.W ∩ q.W ≠ ∅ do
3      O+_{s.W} ← ∅
4      for each w ∈ s.W do
5          o ← GET-NEXT-NEAREST-OBJECT(w, s)
6          while (|O+_{s.W}| < k) ∧ (dist(o, s) < dist(q, s))
           do
7              O+_{s.W} ← O+_{s.W} ∪ {o}
8              o ← GET-NEXT-NEAREST-OBJECT(w, s)
9          if |O+_{s.W}| ≥ k then
10             break
11     if |O+_{s.W}| < k then
12         RT_q ← RT_q ∪ {s}
```

PROOF (SKETCH). Assume case (i). This case means that $\epsilon \cdot dist(o^*, s) < dist(q, s)$, so, from condition (3) of Definition 3, s cannot be in ART_q. The result of case (ii) is derived from the same approach. □

Now we see that adding users having case (ii) in Theorem 1 into ART_q does not violate Definition 3. To reflect this, the condition of line 6 of Algorithm 1 is replaced with "$(|O^+_{s.W}| < k) \wedge (\epsilon \cdot dist(o, s) < dist(q, s))$".

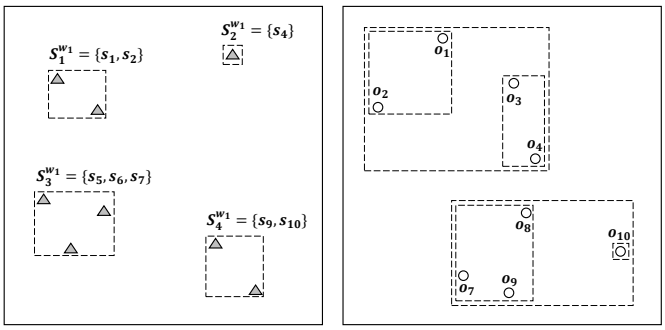
III. PART

Although the baseline algorithm skips unnecessary computation by using the decision version of top-k spatial-keyword searches, it still has two drawbacks. The first one is to *independently* evaluate each user in S. The other is to compute k nearest neighbors *exactly*.

We alleviate the first drawback by using the observation that, if users s_i and s_j, where $w \in s_i.W, s_j.W$, are close, their top-k spatial-keyword objects would be similar. In addition, we overcome the second drawback by exploiting the fact that $q \notin T_s$ if there are at least k spatial-keyword objects being closer to a user s than q. This fact implies that we do not have to care about "nearest" neighbors. From these ideas, we run the decision version of a top-k spatial-keyword search (i) for such close users at the same time and (ii) in a more efficient manner.

A. Indexing

To implement the above ideas, we maintain the users in S based on their locations and keywords. Let S^w be a set of users $s \in S$ such that $w \in s.W$. By using quadtree-like space partitioning, we partition S^w into disjoint subsets so that each subset S^w_i contains close users. Note that S^w_i maintains the minimum bounding rectangle (MBR), $S^w_i.mbr$,



(a) User index (b) Spatial-keyword object index

Fig. 2. Examples of indexing user and spatial-keyword object having keyword w_1 in Fig. 1. Dashed rectangles in (a) and (b) respectively show minimum bounding rectangles and nodes.

which encloses the locations of the users in S_i^w. We repeat this for each keyword appearing in $\bigcup_{s \in S} s.W$. As we explain later, we evaluate users in S_i^w at the same time to enable computation sharing.

Similar to the baseline algorithm, O is indexed by a tree structure. Let O^w be a set of spatial-keyword objects $o \in O$ such that $w \in o.W$, and O^w is indexed by an R-tree. We build such an R-tree for each keyword appearing in $\bigcup_{o \in O} o.W$.

EXAMPLE 2. *Fig. 2 depicts our indices for a keyword w_1, where users and spatial-keywords objects follow Fig. 1. Figs. 2(a) and 2(b) show users and spatial-keyword objects having w_1, respectively. For example, $S^{w_1} = \{s_1, s_2, s_4, s_5, s_6, s_7, s_9, s_{10}\}$, and this is partitioned into four disjoint subsets $S_1^{w_1}$, $S_2^{w_1}$, $S_3^{w_1}$, and $S_4^{w_1}$. The MBR of each subset is illustrated by a dashed rectangle in Fig. 2(a).*

B. Filtering

Given an ART query q, we consider S^w where $w \in q.W$. Now let us focus on a subset S_i^w of S^w. We find a set O^+ of spatial-keyword objects that are closer to s than q for all $s \in S_i^w$. In other words, for O^+, we have $\forall s \in S_i^w$, $\forall o \in O^+$, $dist(o, s) < dist(q, s)$. Notice that if $|O^+| \geq k$, we can ignore all users in S_i^w from Definition 3.

Thanks to the R-tree structure and the MBR of each S_i^w, we can compute lower- and upper-bound distances between S_i^w and a node of an R-tree. Let $ldist(\cdot, \cdot)$ and $udist(\cdot, \cdot)$ represent lower- and upper-bound distances, respectively[3]. For example, $udist(q, S_i^w)$ represents an upper-bound distance between $q.l$ and $S_i^w.mbr$. In addition, let O_n^w be a set of objects maintained by the sub-tree rooted at a node n of the R-tree for a keyword w. We derive the following theorems from our index structures.

THEOREM 2. *Consider a keyword $w \in q.W$ and a node n of an R-tree for w. If $udist(q, S_i^w) < \epsilon \cdot ldist(n, S_i^w)$, we can prune the sub-tree rooted at n without violating the correctness of ART_q.*

[3]A lower-bound (upper-bound) distance between two rectangles is their shortest (longest) distance.

Algorithm 2: PART

Input: O, S, and q
Output: ART_q

```
1 ART_q ← ∅
2 for each w ∈ q.W do
3     for each S_i^w ∈ S^w do
4         ⟨N, O+⟩ ← GET-CANDIDATES(q, w, S_i^w)
5         if |O+| < k then
6             VERIFY(q, ART_q, w, S_i^w, N, O+)
```

PROOF. We have $ldist(n, S_i^w) \leq dist(o, s)$ for every $s \in S_i^w$ and every $o \in O_n^w$. Also, we have $dist(q, s) \leq udist(q, S_i^w)$ for every $s \in S_i^w$. Therefore, if $udist(q, S_i^w) < \epsilon \cdot ldist(n, S_i^w)$, we have $dist(q, s) < \epsilon \cdot dist(o, s)$ for every $s \in S_i^w$ and every $o \in O_n^w$. We then see that, in this case, each $o \in O_n^w$ does not affect the answer to Problem 3 for each $s \in S_i^w$. Consequently, we can ignore O_n^w. □

THEOREM 3. *If $udist(n, S_i^w) < ldist(q, S_i^w)$, we have $o \in O^+$ for every $o \in O_n^w$.*

PROOF. We have $dist(o, s) \leq udist(n, S_i^w)$ for every $s \in S_i^w$ and for every $o \in O_n^w$. Also, we have $ldist(q, S_i^w) \leq dist(q, s)$ for every $s \in S_i^w$. From these facts, if $udist(n, S_i^w) < ldist(q, S_i^w)$, we have $dist(o, s) \leq dist(q, s)$ for every $s \in S_i^w$ and for every $o \in O_n^w$. This theorem hence holds. □

From this theorem, we can add all $o \in O_n^w$ into O^+, thereby we do not need distance computations for the spatial-keyword objects in the sub-tree rooted at n. In addition, the above theorems yield computation sharing among the users in S_i^w. We exploit these filtering techniques in our algorithm.

C. Algorithm Description

Based on the techniques presented in Sections III-A and III-B, we propose PART (Algorithm 2). Given $w \in q.W$, PART conducts the following for each $S_i^w \in S^w$.

1) PART obtains an object set O^+ and a set N of the R-tree for w, where $O^+ = \{o \,|\, w \in o.W, dist(o, S_i^w) < dist(q, S_i^w)\}$ and $N = \{n \,|\, ldist(n, S_i^w) < dist(q, S_i^w)\}$ through the function GET-CANDIDATES.
2) After that, only when $|O^+| < k$, PART computes users $\in S_i^w$ that can be in ART_q, through the function VERIFY.

This is repeated for each $w \in q.W$. We below elaborate on the functions GET-CANDIDATES and VERIFY.

GET-CANDIDATES is described in Algorithm 3. Given a keyword w and q, this function retrieves (i) a set N of leaf nodes n of an R-tree for w such that $ldist(n, S_i^w) < udist(q, S_i^w)$ and (ii) a set O^+ of spatial-keyword objects o such that $dist(o, S_i^w) < dist(q, S_i^w)$.

From the root node of the R-tree for w, we access its nodes by using a heap H.

- If a given node is a leaf node, we add it into N.
- If a given node is an intermediate node, we evaluate each of its child nodes n'.

Algorithm 3: GET-CANDIDATES

Input: q, w, S_i^w, and an R-tree of O for w
Output: $\langle N, O^+ \rangle$

1 $N \leftarrow \varnothing$
2 $O^+ \leftarrow \varnothing$
3 $H \leftarrow$ the root of the R-tree
4 **while** $H \neq \varnothing$ **do**
5 $n \leftarrow$ the front of H
6 Pop the front of H
7 **if** n *is a leaf node* **then**
8 $N \leftarrow N \cup \{n\}$
9 **else**
10 **for** *each child node* n' *of* n **do**
11 **if** $udist(n', S_i^w) < ldist(q, S_i^w)$ **then**
12 $O^+ \leftarrow O^+ \cup O_{n'}$
13 **if** $|O^+| > k$ **then**
14 return $\langle N, O^+ \rangle$
15 **else**
16 **if** $\epsilon \cdot ldist(n', S_i^w) \leq udist(q, S_i^w)$ **then**
17 Push n' into H

Algorithm 4: VERIFY

Input: q, ART_q, w, S_i^w, O^+, and N
Output: ART_q

1 **for** *each* $s \in S_i^w$ **do**
2 $O_s^+ \leftarrow O^+$
3 $N_s \leftarrow \varnothing$
4 **for** *each* $n \in N$ **do**
5 **if** $udist(s, n) < ldist(s, q)$ **then**
6 $O_s^+ \leftarrow O_s^+ \cup O_n$
7 **else**
8 **if** $\epsilon \cdot ldist(s, n) < udist(s, q)$ **then**
9 $N_s \leftarrow N_s \cup \{n\}$
10 **for** *each* $w' \in s.W$ *such that* $w \neq w'$ **do**
11 **if** $|O_s^+| \geq k$ **then**
12 **break**
13 $\langle O_s^+, N_s \rangle \leftarrow \langle O_s^+, N_s \rangle \cup$ GET-CANDIDATES(q, w', s)
14 **if** $|O_s^+| < k$ **then**
15 $ART_q \leftarrow ART_q \cup$ CHECK-USERS(q, s, O_s^+, N_s)

- From Theorem 3, we have $dist(o, S_i^w) < dist(q, S_i^w)$ for all o in O_n^w, if $udist(n', S_i^w) < ldist(q, S_i^w)$. In this case, we add them into O^+. Note that this function terminates whenever we have $|O^+| \geq k$.
- Else, we add n' into H iff $\epsilon \cdot ldist(n', S_i^w) \leq udist(q, S_i^w)$. This is because, from Theorem 2, all objects in O_n^w are not necessary for solving Problem 3, thus can be pruned if $\epsilon \cdot ldist(n', S_i^w) > udist(q, S_i^w)$.

VERIFY is described in Algorithm 4. Given S_i^w, this function retrieves all users $s \in S_i^w$ such that $s \in ART_q$. Specifically, for each $s \in S_i^w$, this function does the following.

1) First, from N and O^+, we compute their specific version to s, i.e., N_s and O_s^+ (lines 2–9).
2) Then, by focusing on keywords in $s.W \backslash \{w\}$, we update N_s and O_s^+ in a similar way to GET-CANDIDATES (lines 10-13). During this, if $|O_s^+| \geq k$, we stop evaluating s, since $q \notin T_s$.
3) After that, if we still have $|O_s^+| < k$, we count the number of objects in O_s^+ and N_s such that $dist(o, s) < dist(q, s)$ through CHECK-USERS. If the count is less than k, it returns s. Otherwise, NULL is returned.

REMARK 1. *Note that PART adds s into ART_q such that $|O_s^+| < k$, and this guarantees correctness. Moreover, increasing ϵ decreases the number of nodes satisfying line 16 of Algorithm 3 and line 8 of Algorithm 4. PART hence guarantees that its computation time becomes shorter by increasing ϵ (if the other parameters are fixed).*

IV. B-PART

In this section, we assume that ART queries are given in a batch. Let Q be a set of ART queries issued at the same time.

Algorithm 5: B-PART

Input: O, S, and Q
Output: ART_q for each $q \in Q$

1 $W_Q \leftarrow \varnothing$
2 $\mathcal{Q} \leftarrow \varnothing$
3 **for** *each* $q \in Q$ **do**
4 $ART_q \leftarrow \varnothing$
5 **for** *each* $w \in q.W$ **do**
6 $W_Q \leftarrow W_Q \cup \{w\}$
7 $\mathcal{Q}[w] \leftarrow \mathcal{Q}[w] \cup \{q\}$
8 **for** *each* $w \in W_Q$ **do**
9 $k_{max} \leftarrow \max_{q \in \mathcal{Q}[w]} q.k$
10 **for** *each* $S_i^w \in S^w$ **do**
11 $\langle N, O^+ \rangle \leftarrow$ BATCH-GET-CANDIDATES$(\mathcal{Q}[w], w, S_i^w)$
12 **if** $(\mathcal{Q}[w] \neq \varnothing) \wedge (|O^+| < k_{max})$ **then**
13 BATCH-VERIFY$(ART_{\mathcal{Q}[w]}, \mathcal{Q}[w], w, S_i^w, N, O^+)$ // $ART_{\mathcal{Q}[w]}$ is a set of ART_q of $q \in \mathcal{Q}[w]$

Assume that ART queries q_1 and q_2 share a keyword w. If we simply run PART for each $q \in Q$, we access S_i^w redundantly (at least twice for q_1 and q_2). Clearly, this approach incurs unnecessary computation, thereby we propose B-PART, which accesses S_i^w only once.

Algorithm description. Algorithm 5 describes B-PART.

1) First, in lines 1–7, to enable batch processing of ART queries having common keywords, we obtain W_Q, which

Algorithm 6: BATCH-GET-CANDIDATES

Input: $\mathcal{Q}[w]$, w, S_i^w, and an R-tree of O for w
Output: $\langle N, O^+ \rangle$

1 Sort $\mathcal{Q}[w]$ in descending order of $udist(q, S_i^w)$
2 $N \leftarrow \varnothing$
3 $O^+ \leftarrow \varnothing$
4 $k_{max} \leftarrow \max_{q \in \mathcal{Q}[w]} q.k$
5 $\epsilon_{min} \leftarrow \min_{q \in \mathcal{Q}[w]} q.\epsilon$
6 $ldist_{min} \leftarrow \min_{q \in \mathcal{Q}[w]} ldist(q, S_i^w)$
7 $q_1 \leftarrow$ the first ART query in $\mathcal{Q}[w]$
8 $H \leftarrow$ the root of the R-tree
9 **while** $H \neq \varnothing$ **do**
10 $n \leftarrow$ the front of H
11 Pop the front of H
12 **if** *n is a leaf node* **then**
13 $N \leftarrow N \cup \{n\}$
14 **else**
15 **for** *each child node n' of n* **do**
16 **if** $udist(n', S_i^w) < ldist_{min}$ **then**
17 $O^+ \leftarrow O^+ \cup O_{n'}$
18 **if** $|O^+| \geq k_{max}$ **then**
19 return $\langle N, O^+ \rangle$
20 **else**
21 **while** $(|O^+ \cup O_{n'}| \geq q_1.k) \wedge (udist(n', S_i^w) < ldist(q_1, S_i^w))$ **do**
22 $\mathcal{Q}[w] \leftarrow \mathcal{Q}[w] \backslash \{q_1\}$
23 **if** $\mathcal{Q}[w] = \varnothing$ **then**
24 return $\langle N, O^+ \rangle$
25 $q_1 \leftarrow$ the first ART query in $\mathcal{Q}[w]$
26 **if** $\epsilon_{min} \cdot ldist(n', S_i^w) \leq udist(q_1, S_i^w)$ **then**
27 Push n' into H

Algorithm 7: BATCH-VERIFY

Input: $ART_{\mathcal{Q}[w]}$, $\mathcal{Q}[w]$, w, S_i^w, O^+, and N

1 $k_{max} \leftarrow \max_{q \in \mathcal{Q}[w]} q.k$
2 $\epsilon_{min} \leftarrow \min_{q \in \mathcal{Q}[w]} q.\epsilon$
3 **for** *each $s \in S_i^w$* **do**
4 $O_s^+ \leftarrow O^+$, $N_s \leftarrow \varnothing$
5 $dist_{min} \leftarrow \min_{q \in \mathcal{Q}[w]} dist(q, s)$
6 $dist_{max} \leftarrow \max_{q \in \mathcal{Q}[w]} dist(q, s)$
7 **for** *each $n \in N$* **do**
8 **if** $udist(s, n) < dist_{min}$ **then**
9 $O_s^+ \leftarrow O_s^+ \cup O_n$
10 **else**
11 **if** $\epsilon_{min} \cdot ldist(s, n) < dist_{max}$ **then**
12 $N_s \leftarrow N_s \cup \{n\}$
13 **for** *each $w' \in s.W$ such that $w \neq w'$* **do**
14 **if** $|O_s^+| \geq k$ **then**
15 **break**
16 $\langle O_s^+, N_s \rangle \leftarrow \langle O_s^+, N_s \rangle \cup$ BATCH-GET-CANDIDATES$(\mathcal{Q}[w], w', s)$
17 **if** $|O_s^+| < k_{max}$ **then**
18 **for** *each $q \in \mathcal{Q}[w]$* **do**
19 $ART_q \leftarrow ART_q \cup$ CHECK-USERS(q, s, O_s^+, N_s)

is a set of keywords appearing in $\bigcup_{q \in Q} q.W$, and $\mathcal{Q}$, which is a collection of $\mathcal{Q}[w]$ (a set of ART queries having keyword w).

2) We then compute ART_q of each $q \in Q$ by using batch processing versions of GET-CANDIDATES and VERIFY, which are presented below, in lines 8–13.

BATCH-GET-CANDIDATES is a variant of GET-CANDIDATES for processing ART queries having a keyword w in a batch. Algorithm 6 details this variant. Given $\mathcal{Q}[w]$ and w, we first sort $\mathcal{Q}[w]$ in descending order of $udist(q, S_i^w)$. Then, we initialize N and O^+ as with GET-CANDIDATES, but different from it, BATCH-GET-CANDIDATES computes k_{max} and ϵ_{min}, i.e., the maximum (minimum) value of k (ϵ) used in $\mathcal{Q}[w]$. We next compute $ldist_{min} = \min_{q \in \mathcal{Q}[w]} ldist(q, S_i^w)$. Let q_1 be the first ART query in $\mathcal{Q}[w]$, and BATCH-GET-CANDIDATES accesses each node n of the R-tree for w in a similar way to GET-CANDIDATES.

- If n is a leaf node, we add it into N.
- If n is an intermediate node, we evaluate each of its child nodes n'.
  - If $udist(n', S_i^w) < ldist_{min}$, $udist(n', S_i^w) < ldist(q, S_i^w)$ holds for every $q \in \mathcal{Q}[w]$. We hence add all spatial-keyword objects in $O_{n'}$ into O^+. In addition, if $|O^+| \geq k_{max}$, all users in S_i^w do not satisfy condition (i) of Definition 3 for all $q \in \mathcal{Q}[w]$, so this function is terminated.
  - Else, we use the observation that ART queries q with large $udist(q, S_i^w)$ tend to have more spatial-keyword objects with closer distances to the users in S_i^w than q. That is, if $|O^+ \cup O_{n'}| \geq q_1.k$ and $udist(n', S_i^w) < ldist(q_1, S_i^w)$, it is guaranteed that there are at least $q_1.k$ spatial-keyword objects being closer to the users in S_i^w than q_1. We therefore have $q_1 \notin T_s$ for every $s \in S_i^w$, so we remove q_1 from $\mathcal{Q}[w]$ and consider the next ART query. After this while-loop (see line 9), we add n' into the heap by using the same rationale as in GET-CANDIDATES.

BATCH-VERIFY is essentially the same as VERIFY. The main difference is to use aggregation values, i.e., k_{max} (line 1), ϵ_{min} (line 2), $dist_{min}$ (line 5), and $dist_{max}$ (line 6), to keep correctness.

V. EXPERIMENT

This section reports our experimental results. All experiments were conducted on a machine with 3.0GHz Intel Xeon Gold 6154 CPU and 512GB RAM.

A. Setting

Algorithms. We evaluated the following algorithms.

- SF-SEP [17]: We applied the algorithm proposed in [17] to our problem. This algorithm maintains O by using an IR-tree [31]. Given an ART query, this algorithm runs top-k spatial-keyword searches for all users in S on the IR-tree.
- RG [32]: Similarly to SF-SEP, we applied the algorithm proposed in [32] to our problem. This algorithm also employs an IR-tree to maintain O[4]. Different from SF-SEP, RG early stops a top-k spatial-keyword search whenever it knows that q cannot be included in T_s.
- BL: Our baseline algorithm presented in Section II-B.
- PART: Our ART query processing algorithm proposed in Section III.
- B-PART: Our batch ART query processing algorithm proposed in Section IV.

These algorithms ran on a single thread.

Datasets. We used the following two real datasets.

- Places[5]: A set of public places inside the United States. Each object has a geo-location and keywords. The number of distinct keywords is approximately 2.1 million, and the average number of keywords held by an object is 5.2.
- Twitter[6]: A set of geo-tagged tweets generated inside the United States. The number of distinct keywords is approximately 0.54 million, and the average number of keywords held by an object is 2.9.

For each dataset, we randomly picked at most 5 million data and used them as O. After that, we randomly picked at most 1 million data and used them as S.

The locations and keywords of ART queries were randomly chosen objects from O. We set k_{max}, and the value of k for an ART query is a random one that follows a uniform distribution in $[1, k_{max}]$. Also, the value of ϵ for an ART query is a random one that follows a normal distribution in $\mathcal{N}(\bar{\epsilon}, 1)$.

Parameter. The default values of k_{max}, $|O|$, $|S|$, $\bar{\epsilon}$, and $|Q|$ were respectively 10, 1000000, 250000, 1.5, and 1000. When we studied the impact of a parameter, the other parameters were fixed by their default values.

B. Result

We generated $|Q|$ ART queries in the way introduced in Section V-A and measured the running time to process Q. (SF-SEP, RG, BL, and PART processed each ART query in Q sequentially.) Each experiment repeated this 100 times, and we report the average result[7].

TABLE II
AVERAGE RUNNING TIME [SEC] OF EACH ALGORITHM

Algorithm	Places	Twitter
SF-SEP	9750.08	8094.60
RG	38.19	30.17
BL	25.76	15.52
PART	1.75	1.45
B-PART	0.86	0.78

Comparison with SF-SEP, RG, and BL. Table II shows the running time of each algorithm to process Q at the default parameter setting[8]. We see that PART and B-PART outperform the other algorithms on both datasets. PART and B-PART are at least one order of magnitude faster than BL, RG, and SF-SEP. In addition, by comparing B-PART with PART, it can be observed that our batch processing functions well (B-PART needed about half the running time of PART).

Next, each algorithm needs longer running time on Places than on Twitter. Spatial-keyword objects and users in Places tend to have similar keywords compared with those in Twitter, since Places has less distinct keywords than Twitter, as described in Section V-A. Due to this fact, each algorithm needs to check more users and spatial-keyword objects in the case of Places.

Last, SF-SEP is clearly outperformed by the other algorithms. From this result, repeating the top-k spatial-keyword search for each user in S is not an appropriate solution, and the importance of employing Problem 3 can be seen. We omit the results of SF-SEP in the subsequent experiments, because of its inferior performance.

Varying k_{max}. Fig. 3 exhibits the results with varying k_{max}. PART and B-PART always outperform the other algorithms. The main result here is the robustness of PART and B-PART against k. On the other hand, BL is sensitive to k. This difference is derived from their data (spatial-keyword objects) access patterns. PART and B-PART simply evaluate whether $dist(q, s)$ is sufficiently small or large to solve the top-k spatial-keyword decision problem, whereas BL incrementally computes k nearest neighbors. The latter approach involves more data accesses until we know whether $q \in T_s$. This observation suggests the effectiveness of our bounding approach.

Varying $|S|$. We next investigated the impact of the number of users, and Fig. 4 shows the result. Although all algorithms need more running time as $|S|$ increases, PART and B-PART are less sensitive to $|S|$ than RG and BL. Actually, RG and BL scale linearly to $|S|$, whereas PART and B-PART scale sub-linearly to $|S|$ in our experiments. As PART and B-PART employ computation sharing among users in S_i^w, they scale better to large S.

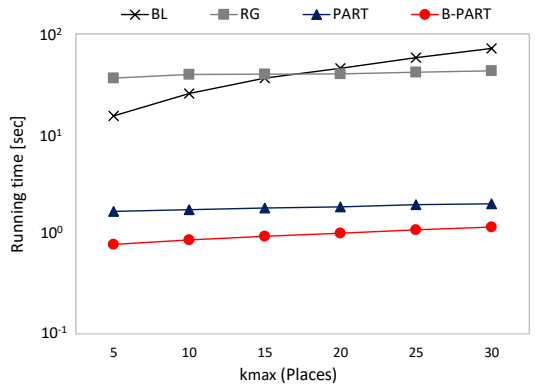
[4] [32] originally assumes a road network and its original index is optimized for the road network. To make their technique available in the Euclidean space, we used IR-tree.

[5] https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces

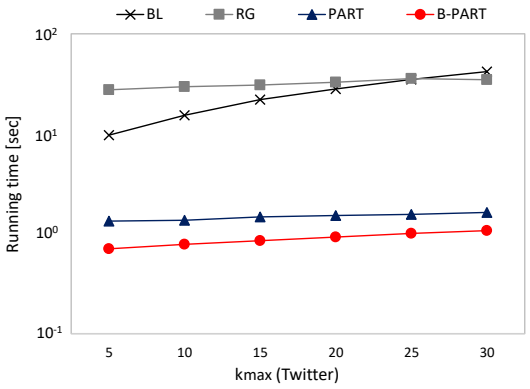
[6] http://www.ntu.edu.sg/home/gaocong/datacode.html

[7] Although this time corresponds to solving Problem 2, the average time to solve Problem 1 is trivially obtained by dividing the time by $|Q|$.

[8] From Table II, we see that the average running time of PART for a single ART query is 1.75 [msec] on Places (since 1.75 [sec] / $|Q|$ = 0.00175 [sec]).

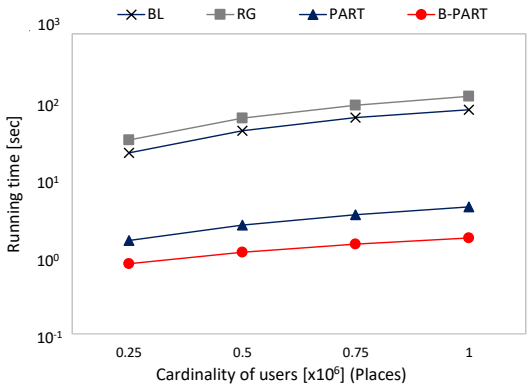


(a) Places

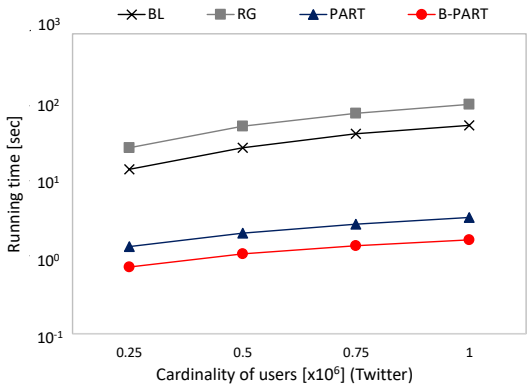


(b) Twitter

Fig. 3. Impact of k_{max}



(a) Places



(b) Twitter

Fig. 4. Impact of $|S|$

Varying $|O|$. The scalability to $|O|$ was also tested, and we report the result in Fig. 5. We observe that PART and B-PART keep short running times: they process 1,000 ART queries within two seconds. Moreover, we see that, compared with the result in Fig. 4, the impact of $|O|$ is smaller, although the running time of each algorithm increases a bit. When $|O|$ is larger, RG has more nodes in the IR-tree, so the time for the top-k spatial-keyword search for each user slightly increases. In the case of BL, the time for GET-NEXT-NEAREST-OBJECT becomes longer for a larger $|O|$. Similarly, PART and B-PART need to access more nodes of R-trees, thus their running times also become a bit longer for a larger $|O|$.

Varying $\bar{\epsilon}$. Fig. 6 shows the experimental results with varying $\bar{\epsilon}$. BL and RG receive the benefit of ϵ a little, whereas the running times of PART and B-PART decrease as $\bar{\epsilon}$ increases. As described in the remark in Section III-C, PART guarantees that its running time decrease as ϵ increases (due to the fact that the number of access nodes decreases). B-PART inherits this advantage.

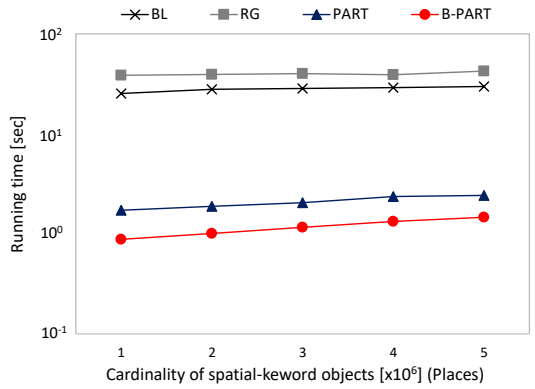
Varying $|Q|$. Last, we studied the impact of the number of ART queries issued at the same time. Fig. 7 shows that the running times of BL, RG, and PART increase linearly to $|Q|$, whereas that of B-PART increases sub-linearly to $|Q|$. This is a reasonable result. BL, RG, and PART process each $q \in Q$ iteratively, thereby their linearity to $|Q|$ is a natural observation. As for B-PART, a larger Q tends to contain more similar ART queries (w.r.t. locations and keywords). B-PART processes such ART queries at the same time, so it scales better than the other algorithms w.r.t. the size of Q.

VI. Related Work

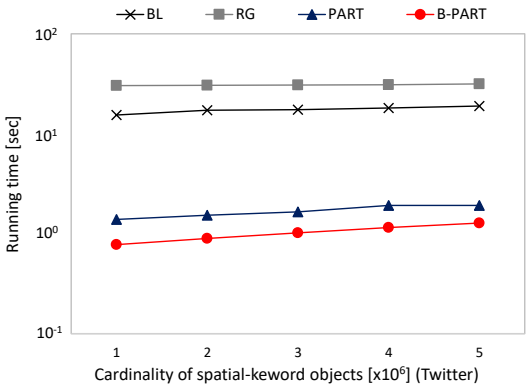
Reverse Top-k and kNN Queries were first considered in multi-dimensional databases [20], [21], [33]–[36]. These works focus only on numeric objects, so their algorithms are not available for our problem. Approximate reverse NN queries were also addressed, e.g., in [22]. However, their solutions cannot deal with kNNs and keywords are not taken into consideration.

Spatial-Keyword Query. Because of the increase in the number of spatial-keyword objects, spatial-keyword databases and querying have been receiving much attention [12]. Many types of spatial-keyword queries have therefore been devised, such as spatial-keyword match queries [23], top-k spatial-keyword queries [19], [27], moving spatial-keyword queries [37], and why-not spatial-keyword queries [38], to name a few. Their variants, e.g., streaming data [9], [15] and distributed computing [26], have also been developed (although these settings are beyond the scope of this paper).

Among these queries, top-k spatial-keyword queries are the most related studies to our problem. As mentioned in Section I, a straightforward approach to processing an ART query is to run a top-k spatial-keyword search for each user in S. Our

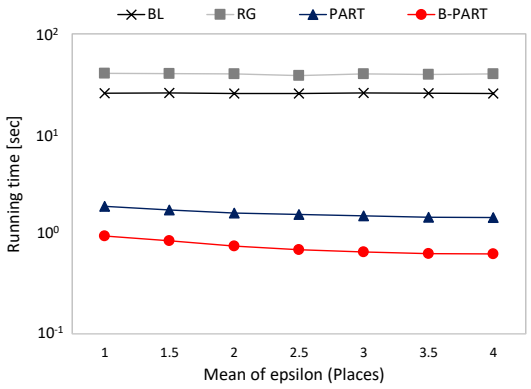


(a) Places

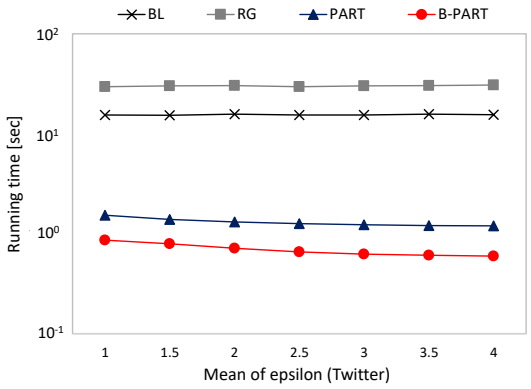


(b) Twitter

Fig. 5. Impact of $|O|$



(a) Places



(b) Twitter

Fig. 6. Impact of $\bar{\epsilon}$

experimental studies demonstrate that this approach incurs a huge computational cost even when we employ a state-of-the-art algorithm [17].

There are some works that consider reverse spatial-keyword queries [32], [39]–[43]. However, [39], [43] consider a totally different problem: they consider the problem of finding a region or location of a point p that can be the top-k spatial-keyword object of the maximum number of users. Although [41], [42] addressed a similar problem, their definition of spatial-keyword relevance is different from ours. Because their techniques are optimized for this definition, employing them and how to deal with ϵ in them are not trivial. [40] assumes a road network, and its proposed techniques hold only on road networks. It hence cannot deal with ART queries. Although [32] also assumes a road network, its framework is available by replacing its index with one for the Euclidean space. Our experimental results confirm that PART is much faster than this variant of [32].

VII. Conclusion

Finding potential users is an important approach in market analysis. In location-based services, there are huge users nowadays, thereby effective market analysis leads to gains. To support finding of potential users in spatial-keyword databases, we proposed a new query, namely approximate reverse top-k spatial-keyword (ART) query. Because processing this query is computationally challenging, we devised an efficient ART query processing algorithm, PART. In addition, to deal with multiple ART queries issued at the same time, we proposed B-PART, which processes such ART queries in a batch. We conducted extensive experiments by using real datasets. The experimental results demonstrate that (i) PART is much faster than baseline algorithms and (ii) B-PART further improves batch ART query processing.

This paper focused on static data and a single machine, because this is the first work for ART query processing. In spatial-keyword databases, stream data [15] and distributed computing [26] receive attention. ART query processing with these settings is an open problem.

Acknowledgements

This research is partially supported by JST PRESTO Grant Number JPMJPR1931 and JST CREST Grant Number JPMJCR21F2.

References

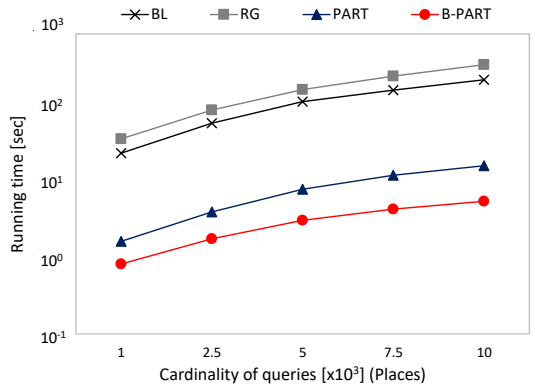
[1] R. Taniguchi, D. Amagata, and T. Hara, "Efficient retrieval of top-k weighted triangles on static and dynamic spatial data," *IEEE Access*, vol. 10, pp. 55 298–55 307, 2022.

[2] D. Amagata and T. Hara, "Monitoring maxrs in spatial data streams." in *EDBT*, 2016, pp. 317–328.

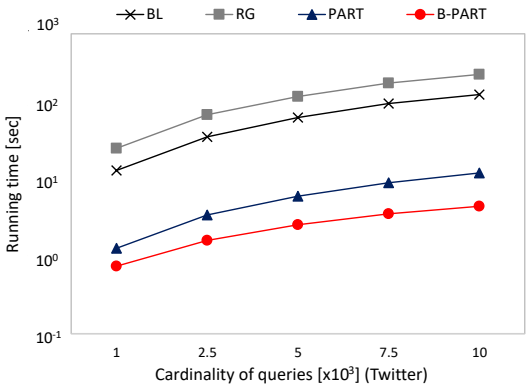
[3] Y. Nakayama, D. Amagata, and T. Hara, "Probabilistic maxrs queries on uncertain data," in *DEXA*, 2017, pp. 111–119.

[4] D. Amagata, Y. Arai, S. Fujita, and T. Hara, "Learned k-nn distance estimation," in *SIGSPATIAL*, 2022, pp. 1–4.

[5] R. Taniguchi, D. Amagata, and T. Hara, "Efficient retrieval of top-k weighted spatial triangles," in *DASFAA*, 2022, pp. 224–231.



(a) Places



(b) Twitter

Fig. 7. Impact of $|Q|$

[6] D. Amagata and T. Hara, "A general framework for maxrs and maxcrs monitoring in spatial data streams," *ACM Transactions on Spatial Algorithms and Systems*, vol. 3, no. 1, pp. 1–34, 2017.

[7] D. Amagata, Y. Sasaki, T. Hara, and S. Nishio, "Probabilistic nearest neighbor query processing on distributed uncertain data," *Distributed and Parallel Databases*, vol. 34, pp. 259–287, 2016.

[8] D. Amagata and T. Hara, "Identifying the most interactive object in spatial databases," in *ICDE*, 2019, pp. 1286–1297.

[9] A. Almaslukh and A. Magdy, "Evaluating spatial-keyword queries on streaming data," in *SIGSPATIAL*, 2018, pp. 209–218.

[10] D. Amagata, T. Hara, and S. Nishio, "Distributed top-k query processing on multi-dimensional data with keywords," in *SSDBM*, 2015, pp. 10:1–10:12.

[11] D. Amagata, S. Tsuruoka, Y. Arai, and T. Hara, "Feat-sksj: Fast and exact algorithm for top-k spatial-keyword similarity join," in *SIGSPATIAL*, 2021, pp. 15–24.

[12] Z. Chen, L. Chen, G. Cong, and C. S. Jensen, "Location-and keyword-based querying of geo-textual data: a survey," *The VLDB Journal*, pp. 1–38, 2021.

[13] S. Nishio, D. Amagata, and T. Hara, "Geo-social keyword top-k data monitoring over sliding window," in *DEXA*, 2017, pp. 409–424.

[14] X. Wang, Y. Zhang, W. Zhang, X. Lin, and Z. Huang, "Skype: Top-k spatial-keyword publish/subscribe over sliding window," *PVLDB*, vol. 9, no. 7, pp. 588–599, 2016.

[15] S. Nishio, D. Amagata, and T. Hara, "Lamps: Location-aware moving top-k pub/sub," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 352–364, 2022.

[16] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *PVLDB*, vol. 2, no. 1, pp. 337–348, 2009.

[17] F. M. Choudhury, J. S. Culpepper, Z. Bao, and T. Sellis, "Batch processing of top-k spatial-textual queries," *ACM Transactions on Spatial Algorithms and Systems*, vol. 3, no. 4, pp. 1–40, 2018.

[18] G. Kalamatianos, G. J. Fakas, and N. Mamoulis, "Proportionality in spatial keyword search," in *SIGMOD*, 2021, pp. 885–897.

[19] C. Zhang, Y. Zhang, W. Zhang, and X. Lin, "Inverted linear quadtree: Efficient top k spatial keyword search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 7, pp. 1706–1721, 2016.

[20] D. Amagata and T. Hara, "Reverse maximum inner product search: How to efficiently find users who would like to buy my item?" in *RecSys*, 2021, pp. 273–281.

[21] A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis, "Branch-and-bound algorithm for reverse top-k queries," in *SIGMOD*, 2013, pp. 481–492.

[22] A. Hidayat, S. Yang, M. A. Cheema, and D. Taniar, "Reverse approximate nearest neighbor queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 2, pp. 339–352, 2017.

[23] A. R. Mahmood, A. Daghistani, A. M. Aly, M. Tang, S. Basalamah, S. Prabhakar, and W. G. Aref, "Adaptive processing of spatial-keyword data over a distributed streaming cluster," in *SIGSPATIAL*, 2018, pp. 219–228.

[24] D. Amagata and T. Hara, "Reverse maximum inner product search: Formulation, algorithms, and analysis," *ACM Transactions on the Web*, 2023.

[25] N. Taguchi, D. Amagata, and T. Hara, "Geo-social keyword skyline queries," in *DEXA*, 2017, pp. 425–435.

[26] S. Tsuruoka, D. Amagata, S. Nishio, and T. Hara, "Distributed spatial-keyword knn monitoring for location-aware pub/sub," in *SIGSPATIAL*, 2020, pp. 111–114.

[27] A. Cary, O. Wolfson, and N. Rishe, "Efficient and scalable method for processing top-k spatial boolean queries," in *SSDBM*, 2010, pp. 87–95.

[28] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: an experimental evaluation," *PVLDB*, vol. 6, no. 3, pp. 217–228, 2013.

[29] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[30] J. Qi, Y. Tao, Y. Chang, and R. Zhang, "Theoretically optimal and empirically efficient r-trees with strong parallelizability," *PVLDB*, vol. 11, no. 5, pp. 621–634, 2018.

[31] Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang, "Ir-tree: An efficient index for geographic document search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 4, pp. 585–599, 2010.

[32] J. Zhao, Y. Gao, G. Chen, C. S. Jensen, R. Chen, and D. Cai, "Reverse top-k geo-social keyword queries in road networks," in *ICDE*, 2017, pp. 387–398.

[33] M. A. Cheema, X. Lin, W. Zhang, and Y. Zhang, "Influence zone: Efficiently processing reverse k nearest neighbors queries," in *ICDE*, 2011, pp. 577–588.

[34] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 201–212, 2000.

[35] Y. Tao, D. Papadias, and X. Lian, "Reverse $k$nn search in arbitrary dimensionality," in *VLDB*, 2004, pp. 744–755.

[36] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg, "Reverse top-k queries," in *ICDE*, 2010, pp. 365–376.

[37] D. Wu, M. L. Yiu, and C. S. Jensen, "Moving spatial keyword queries: Formulation, methods, and analysis," *ACM Transactions on Database Systems*, vol. 38, no. 1, pp. 1–47, 2013.

[38] L. Chen, J. Xu, X. Lin, C. S. Jensen, and H. Hu, "Answering why-not spatial keyword top-k queries via keyword adaption," in *ICDE*, 2016, pp. 697–708.

[39] F. M. Choudhury, J. S. Culpepper, T. Sellis, and X. Cao, "Maximizing bichromatic reverse spatial and textual k nearest neighbor queries," *PVLDB*, vol. 9, no. 6, pp. 456–467, 2016.

[40] Y. Gao, X. Qin, B. Zheng, and G. Chen, "Efficient reverse top-k boolean spatial keyword queries on road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 5, pp. 1205–1218, 2014.

[41] J. Lu, Y. Lu, and G. Cong, "Reverse spatial and textual k nearest neighbor search," in *SIGMOD*, 2011, pp. 349–360.

[42] Y. Lu, J. Lu, G. Cong, W. Wu, and C. Shahabi, "Efficient algorithms and cost models for reverse spatial-keyword k-nearest neighbor search," *ACM Transactions on Database Systems*, vol. 39, no. 2, pp. 1–46, 2014.

[43] P. Zhao, H. Fang, V. S. Sheng, Z. Li, J. Xu, J. Wu, and Z. Cui, "Monochromatic and bichromatic ranked reverse boolean spatial keyword nearest neighbors search," *World Wide Web*, vol. 20, no. 1, pp. 39–59, 2017.