Lamps: Location-Aware Moving Top-k Pub/Sub

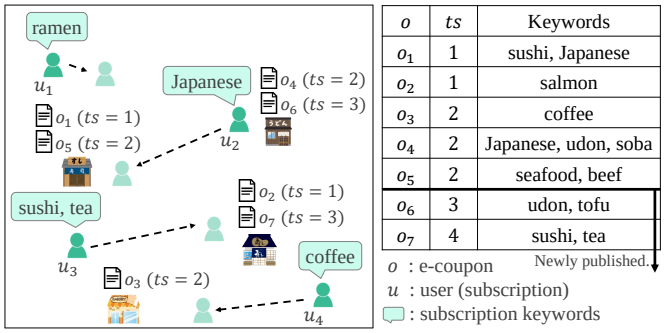
Shunya Nishio, Daichi Amagata, Takahiro Hara
Osaka University, Japan
Email: {nishio.syunya, amagata.daichi, hara}@ist.osaka-u.ac.jp

***Abstract*—Huge amounts of spatio-textual objects, such as geo-tagged tweets, are being generated at an unprecedented scale, leading to a variety of applications such as location-based recommendation and sponsored search. Many of these applications need to support moving top-k spatio-textual subscriptions. For example, while walking, a tourist issues a moving subscription and looks for top-k advertisements published by nearby shops. Unfortunately, existing methods that monitor the results of spatio-textual subscriptions support only static top-k subscriptions or moving boolean subscriptions.**

In this paper, we propose a novel system, called Lamps (Location-Aware Moving Top-k Pub/Sub), which continuously monitors the top-k most relevant spatio-textual objects for a large number of moving top-k spatio-textual subscriptions simultaneously. To the best of our knowledge, this is the first study of a location-aware moving top-k pub/sub system. As with existing works on continuous moving top-k subscription processing, Lamps employs the concept of a safe region to monitor top-k results. However, unlike with existing works that assume static objects, top-k result updates may be triggered by newly generated objects. To continuously monitor the top-k results for massive moving subscriptions efficiently, we propose SQ-Tree, a novel index based on safe regions, to filter subscriptions whose top-k results do not change. Moreover, to reduce the expensive cost of safe region re-evaluation, we develop a novel approximation technique for safe region construction. Our experimental results on real datasets show that Lamps achieves higher performance than baseline approaches.

I. Introduction

Because of the prevalence of social networks and GPS-enabled mobile devices, huge amounts of objects with both spatial and textual information, referred to as spatio-textual objects, have been generated in a streaming fashion. This has led to the popularity of location-aware pub/sub systems in many applications, such as location-based recommendation [15] and sponsored search [11]. In such a system, a large number of users register their interests (e.g., keywords and locations) as spatio-textual subscriptions. Then, the system delivers a stream of spatio-textual objects (e.g., geo-tagged tweets, e-coupons, and advertisements) generated by publishers (e.g., restaurants) to the relevant subscriptions. Various location-aware pub/sub systems have been proposed [1], [2], [8], [19], [22]. Most of them focus on boolean matching, thereby users may receive few matching objects or may be overwhelmed by a huge volume of matching objects. In such a situation, a top-k subscription, which can control the size of the objects to be received, is useful. Unfortunately, existing works focus on static subscriptions and cannot support moving subscriptions efficiently. Note that many real-world applications need to support moving subscriptions [7], [8], [10], [12], [23]. For example, while walking, a tourist looks for top-k advertisements published by nearby shops. Because he/she is walking, his/her location (*subscription location*) changes continuously.



o	ts	Keywords
o_1	1	sushi, Japanese
o_2	1	salmon
o_3	2	coffee
o_4	2	Japanese, udon, soba
o_5	2	seafood, beef
o_6	3	udon, tofu
o_7	4	sushi, tea

Fig. 1: A location-aware moving top-k pub/sub system. At $ts = 3$, o_6 and o_7 are newly published and each user moves to the tip of the arrow.

Motivated by the fact that none of the current systems can support moving top-k subscriptions against streaming objects, in this paper, we propose a novel system, called Lamps (**L**ocation-**A**ware **M**oving Top-k **P**ub/**S**ub), which continuously monitors the top-k most relevant spatio-textual objects for a large number of moving top-k spatio-textual subscriptions simultaneously. Specifically, for each subscription, we score objects based on their spatial and textual similarities, and the top-k results are continuously maintained against streaming objects (i.e., object generations and expirations) and the movements of users.

Example 1. *Fig. 1 shows an example of a location-aware moving top-k pub/sub system that delivers e-coupons to potential consumers. In this example, there are four users (subscriptions) and four restaurants (publishers) that continuously generate e-coupons. Each restaurant can generate multiple e-coupons and delete its generated e-coupons. Each user registers his/her interest as a subscription and monitors the top-1 most relevant e-coupon. At timestamp $ts = 2$, we assume that five e-coupons (o_1 - o_5) have been published and the top-1 result for user u_3 is e-coupon o_1, because o_1 has the highest spatial and textual similarities to the subscription of u_1. In other words, o_1 is the closest to the location of u_3 and has the common keyword "sushi" as u_3. Also, the top-1 result for u_2 is o_4. At $ts = 3$, new e-coupons o_6 and o_7 are published and each user moves, and then the result for each user is updated. The top-1 result for u_3 becomes o_7, because o_7 is closer to the current location of u_3 and its keywords are more similar to the keywords of u_3. Furthermore, u_2 moves*

west, thereby the top-1 result for u_2 becomes o_1 because o_1 is closer to u_2 than o_4.

Challenges. There are two key challenges for efficiently monitoring the top-k result of each moving subscription.

The first challenge is to monitor the up-to-date top-k results for a massive number of subscriptions over a stream of spatio-textual objects. A straightforward approach works as follows: When a new spatio-textual object o is generated, for each subscription s, we calculate the score of o. If the score of o is better than the score of the current k-th result, o becomes the result of s. Moreover, when an object o' expires, for each subscription s whose top-k result contains o', we need to re-evaluate its result. For example, in Fig. 1, the expiration of o_1 invalidates the current top-1 result of u_2, thereby we have to re-evaluate the result for u_2. The straightforward approach accesses all objects and calculates the scores, but this approach is computationally expensive if the number of subscriptions is large and the spatio-textual objects are frequently generated and frequently expire.

The second challenge is to monitor the up-to-date top-k result for each subscription against the movements of users. When a user u moves, the score of each object for u varies, so the top-k result of u may change. A straightforward approach, which re-calculates the scores of all objects and updates the top-k result of u whenever u moves, incurs an expensive computational cost. To avoid unnecessary computation, several works have proposed safe region techniques for a moving top-k subscription [10], [16], [23]. A safe region is a region where the top-k result does not change. In other words, the top-k result needs to be re-evaluated only when the user exits the safe region. These safe regions however fail to function when streaming objects are considered. This is because safe regions may change when objects are newly generated. When users exit their safe regions or their top-k results change, moreover, we need to re-evaluate their safe regions. Even a state-of-the-art safe region construction algorithm [10] incurs a large evaluation cost if the system stores a lot of objects.

Overview of our proposed system. Lamps employs a novel index, namely SQ-tree, which integrates a Quad-tree with safe regions to effectively maintain continuous moving spatio-textual subscriptions. SQ-tree can filter subscriptions whose top-k results and safe regions do not change when a new object is generated. Besides, we propose an efficient algorithm to retrieve objects that become top-k objects when an object expires or users exit their safe regions. To reduce the expensive safe region evaluation cost, furthermore, we develop a novel approximation technique for safe region construction.

Contributions. We summarize our contributions below.

- We address, for the first time, the problem of monitoring top-k spatio-textual objects for a large number of moving top-k spatio-textual subscriptions. To tackle this problem, we propose a novel system, called Lamps.
- We develop a novel approximation technique for safe region construction to reduce the safe region evaluation cost.
- We propose a novel index based on a Quad-tree and safe regions, SQ-tree, to efficiently filter moving subscriptions whose top-k results and safe regions do not change when a new object is generated.
- We conduct experiments using real datasets, and the results show that Lamps achieves higher performance than baseline approaches.

Organization. We provide preliminary information in Section II. We present Lamps from Section III to Section VI and introduce our experimental results in Section VII. We review some related works in Section VIII and conclude this paper in Section IX.

II. Preliminary

First, in Section II-A we formally present some concepts that are used throughout this paper. Section II-B introduces the safe region technique for a moving top-k subscription.

A. Definition

We define the spatio-textual object and the **M**oving top-**k** **S**patio-**T**extual (MkST) subscription issued by a user.

Definition 1 (Spatio-textual object). *A spatio-textual object, which is generated by a publisher, is defined as $o = (p, t)$, where $o.p$ is the location of o and $o.t$ is a set of keywords of o.*

Definition 2 (MkST subscription). *A MkST subscription is defined as $s = (p, t, k, \alpha)$, where $s.p$ is the current location of the user, $s.t$ is a set of keywords such that $1 \leq |s.t| \leq t_{max}$, $s.k$ is the number of objects requested, and $s.\alpha$ is the preference parameter ($0 < s.\alpha < 1$) used in the scoring function to evaluate the relevance between s and an object. Given an object o and s, the score of o for s, $score(s, o)$, is calculated as follows [10], [20]:*

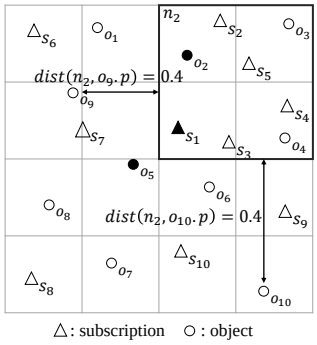
$$score(s, o) = s.\alpha \cdot dist(s.p, o.p) + (1 - s.\alpha) \cdot text(s.t, o.t) \quad (1)$$

where $dist(s.p, o.p)$ is the spatial proximity and $text(s.t, o.t)$ is the textual similarity between s and o. Given an object set O, the answer of s is a subset of O, A, such that (1) $|A| = k$, and (2) $\forall o^* \in A$, $\forall o' \in O \backslash A$, $score(s, o^*) \leq score(s, o')$[1].

In the following of this paper, we abbreviate $s.k$ and $s.\alpha$ as k and α, respectively, if there is no ambiguity.

To compute spatial proximity, we utilize the Euclidean distance $dist(s.p, o.p) = \frac{Edist(s.p, o.p)}{Maxdist}$. Note that $Edist(s.p, o.p)$ is the Euclidean distance between $s.p$ and $o.p$, and $Maxdist$ is the maximum distance in the spatial area $\mathbb{R}^2$ where subscriptions and objects exist. We see that $dist(\cdot, \cdot) \in [0, 1]$. For textual similarity, there are three widely used set-based similarity functions, namely Jaccard, Dice, and Cosine similarities [5]. In this paper, we use Jaccard similarity, i.e., $text(s.t, o.t) = 1 - \frac{|s.t \cap o.t|}{|s.t \cup o.t|}$. Note that in Section VII-C we verify the performance of Lamps when Dice and Cosine similarities are employed. Then, $score(s, o)$ is with in $[0, 1]$.

[1] As with [2], [20], to guarantee the top-k results are textual-relevant, an object must contain at least one common keyword with a subscription to become its top-k result.



s	Keywords
s_1	w_1, w_2
s_2	w_3
s_3	w_2, w_4
s_4	w_3, w_4, w_5, w_6
s_5	w_2, w_5
s_6	w_1, w_5
s_7	w_2, w_4, w_5
s_8	w_1
s_9	w_2, w_3
s_{10}	w_7

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

Newly generated..

Fig. 2: Running example. eight objects $\{o_1, \cdots, o_8\}$ have been generated and two objects o_9 and o_{10} are newly generated.

As with [20], [21], we assume that subscriptions are pre-registered in our system and the number of subscriptions does not change. In Section VI, we will discuss the procedure when a subscription is newly issued or deleted.

Problem statement. We consider a set of MkST subscriptions S and a dynamic set O of spatio-textual objects. Subscribers can move randomly at any time (i.e., movements of users) [22]. Publishers can insert their newly generated objects into O (i.e., object generations) and remove objects that they have generated from O (i.e., object expirations). We aim to continuously monitor the top-k results for all subscriptions against object generations, object expirations, and movements of users.

EXAMPLE 2. *Fig. 2 shows a running example used throughout this paper. In this example, there are ten registered subscriptions $\{s_1, \cdots, s_{10}\}$ and eight objects have been generated $\{o_1, \cdots, o_8\}$. Moreover, two objects o_9 and o_{10} are newly generated, and we assume $s_1.k = 2$. Specifically, o_2 and o_5 have high spatial and textual similarities to s_1. Thus, the top-k results of s_1 are o_2 and o_5.*

B. Safe region technique

Lamps employs the concept of the safe region [16] to monitor top-k results. Therefore, we first define safe region and a relevant concept, the dominant region.

DEFINITION 3 (SAFE REGION). *Given O, $s = (p, t, k, \alpha)$, and A of s, the safe region of s, R, is:*

$$R = \{p' | \forall o^* \in A, \forall o' \in O \backslash A, score(s', o^*) \leq score(s', o')\} \quad (2)$$

where s' is s after moving to a new location p', i.e., $s' = (p', t, k, \alpha)$.

Note that the safe region of s is a region where the top-k result of s does not change.

DEFINITION 4 (DOMINANT REGION). *Given $s = (p, t, k, \alpha)$ and two objects o^* and o', the dominant region of o^* to o', $D_{o^*,o'}$, is:*

$$D_{o^*,o'} = \{p' | score(s', o^*) \leq score(s', o')\} \quad (3)$$

where s' is s after moving to a new location p', i.e., $s' = (p', t, k, \alpha)$.

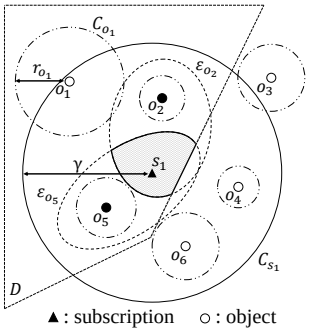


Fig. 3: Example of LSR. Because $s_1.k = 2$ and the top-k results of s_1 are o_2 and o_5, the LSR of s_1 is the shaded region $\mathcal{E}_{o_2} \cap \mathcal{E}_{o_5} \cap D$.

We see that the dominant region of o^* to o' is a region where $score(s, o^*) \leq score(s, o')$. Literature [10] shows the following lemma.

LEMMA 1. *Given O, s, and A, the safe region of s, R, is $R = \cap_{o^* \in A}(\cap_{o' \in O \backslash A} D_{o^*,o'})$.*

Local safe region. To efficiently compute a safe region, literature [10] proposed a local safe region (LSR), which is a subset of the safe region. Notice that we can monitor the exact top-k results by re-evaluating them only when users exit their LSR. [10] models each object o as a circle C_o with a center $o.p$ and radius of $r_o = \frac{1-\alpha}{\alpha} \cdot text(s.t, o.t)$. Then, $score(s, o) = \alpha(dist(s.p, o.p) + r_o)$. The LSR is computed in the following three steps.

Step 1. For each object $o^* \in A$, we compute an ellipse $\mathcal{E}_{o^*}$ and the intersection of these ellipses $\mathcal{E} = \cap_{o^* \in A} \mathcal{E}_{o^*}$. Note that

$$\mathcal{E}_{o^*} = \{p' | dist(s'.p', o^*.p) + dist(s.p, s'.p') \leq \gamma - r_{o^*}\}, \quad (4)$$

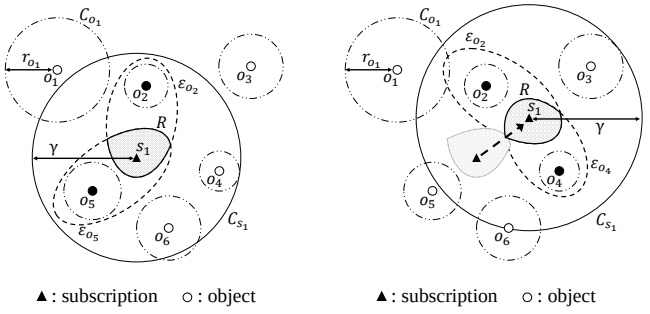
where $\gamma = \max\{dist(s.p, o^*.p) + r_{o^*} | o^* \in A\} + \Delta$ and Δ is a parameter of an approximate ratio. Obviously $\mathcal{E}_{o^*}$ is an ellipse with $s.p$ and $o^*.p$ as two foci.

EXAMPLE 3. *In Fig. 3, because $s_1.k = 2$ and the top-k results of s_1 are o_2 and o_5, $\mathcal{E}$ is the intersection of $\mathcal{E}_{o_2}$ and $\mathcal{E}_{o_5}$.*

Step 2. Let C_s be a circle centered at $s.p$ with radius γ. When $s'.p' \in \mathcal{E}$, each object o does not become the top-k result of s' if C_o is not inside C_s. If C_o is not inside C_s, $\gamma \leq dist(s.p, o.p) + r_o$. Then, $score(s', o^*) < score(s', o)$ for each object $o^* \in A$. Let O_s be the set of objects whose circles are inside C_s and that are not the top-k result of s. Because each object $o' \in O_s$ may become the top-k result of s', we need to compute the region where the top-k result does not change by o'. We compute the dominant region $D_{o^*,o'}$ for each object $o' \in O_s$, and the intersection of these dominant regions $D = \cap_{o^* \in A}(\cap_{o' \in O_s} D_{o^*,o'})$.

EXAMPLE 4. *In Fig. 3, because C_{o_1} and C_{o_3} are not inside C_{s_1}, o_1 and o_3 never become the top-k result of s'_1. O_{s_1} contains only o_4 and o_6, therefore, we compute the dominant regions for o_4 and o_6, and the intersection of these regions.*

Step 3. We compute the LSR $\mathcal{E} \cap D$.



(a) ts (b) $ts+1$

Fig. 4: Example of an ASR. At ts, the ASR of s_1 is the shaded region $\mathcal{E}_{o_2} \cap \mathcal{E}_{o_5}$. At $ts+1$, because the top-k results become o_2 and o_4, the ASR of s_1 is the shaded region $\mathcal{E}_{o_2} \cap \mathcal{E}_{o_4}$.

THEOREM 1. *The total time complexity of the LSR construction algorithm is $\mathcal{O}(Ck|O_s|)$, where C is the cost of calculating the intersection of two polygons (dominant regions).*

PROOF. The first step requires at most $\mathcal{O}(k^2)$ time, because we need to compute k ellipses and their intersection. Given two polygons whose numbers of vertices are v_1 and v_2, $\mathcal{O}((v+l)\log_2(v+l))$ time is required to calculate their intersection points, where $v = v_1 + v_2$ and l is the number of intersection points between the polygon edges [24]. Therefore, the second step requires $\mathcal{O}(Ck|O_s|)$ time, where $C = (v+l)\log_2(v+l)$. The final step incurs $\mathcal{O}(C)$ time. Then the theorem holds. □

Approximate safe region. The local safe region construction needs to compute the dominant region $D_{o^*,o'}$ for each object $o' \in O_s$. Many objects exist in O_s if the system stores a lot of objects, thereby it incurs a large evaluation cost. To avoid this cost, We propose a novel technique that can compute an approximate safe region (ASR), regardless of the distribution of objects. As with LSR, ASR is a subset (partial area) of the exact safe region. Therefore, we can monitor the exact top-k results by re-evaluating them only when users exit thier ASR. Let o_{k+1} be the object whose score for s is the $(k+1)$-th smallest when the safe region of s is computed. In this technique, we set $\gamma = dist(s.p, o_{k+1}.p) + r_{o_{k+1}}$. Then we have the following lemma:

LEMMA 2. $O_s = \varnothing$

PROOF. Because $score(s, o_{k+1}) \leq score(s, o')$ for each object $o' \in O \backslash A$, $\frac{score(s,o_{k+1})}{\alpha} = dist(s.p, o_{k+1}.p) + r_{o_{k+1}} \leq \frac{score(s,o')}{\alpha} = dist(s.p, o'.p) + r_{o'}$. Therefore, $C_{o'}$ is not inside C_s. Then the lemma holds. □

From Lemma 2, we do not need to compute the dominant regions. Therefore, we can compute the safe region only by computing the ellipses and their intersection. That is, our proposed technique can compute an ASR while reducing the evaluation cost. Moreover, ASR is not necessarily a subset of the the LSR. For example, if $\max\{dist(s.p, o^*.p) + r_{o^*} | o^* \in A\} + \Delta < dist(s.p, o_{k+1}.p) + r_{o_{k+1}}$, the ASR is a superset of LSR. To compute the ASR quickly, we find o_{k+1} at the same time as we update the top-k result, which is shown later. In the following part of this paper, when we refer to a safe region, it means its ASR if there is no ambiguity. That is, $R = \mathcal{E}$.

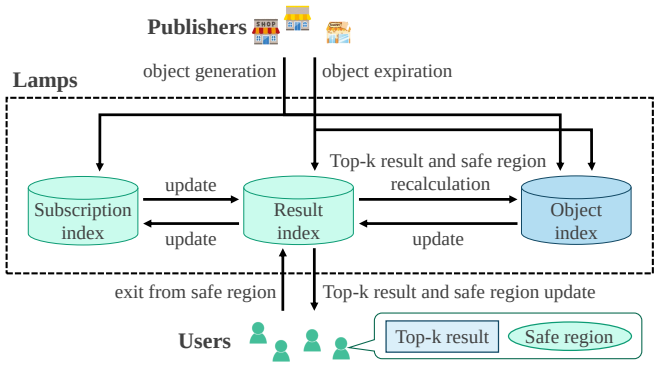


Fig. 5: Framework of Lamps

EXAMPLE 5. *Fig. 4 shows an example of an ASR of s_1. At timestamp ts, the top-k results of s_1 are o_2 and o_5, and o_{k+1} is o_4. Therefore, we compute $s_1.R$ with $\gamma = dist(s_1.p, o_4.p) + r_{o_4}$. At $ts+1$, we need to re-compute $s_1.R$ because s_1 exits $s_1.R$. Because the score of each object changes, the top-k results become o_2 and o_4, and o_{k+1} is o_3. Therefore, we re-compute $s_1.R$ based on o_2, o_4, and o_3.*

THEOREM 2. *The worst time complexity of the ASR construction algorithm is $\mathcal{O}(k^2) \ll \mathcal{O}(ck|O_s|)$.*

PROOF. The ASR construction algorithm only computes k ellipses and their intersection. Then, the theorem holds. □

III. FRAMEWORK

Fig. 5 shows the framework of Lamps. The input of Lamps includes object generations/expirations and movements of users. Here, we introduce how to process them at a high level. Note that, to efficiently deal with them, Lamps employs subscription, result, and object indices.

Object generation. When a new object o is generated, we need to identify the set of subscriptions whose top-k results change because of o. First, we find a candidate set of subscriptions from the subscription index and update the object index. For each subscription in the set, we update the top-k result and its safe region. If the top-k result and the safe region change, they are sent to the user, and the subscription and the result indices are also updated.

Object expiration. When an object o' expires, we need to re-evaluate the results of the subscriptions whose top-k results contain o'. First, we obtain a set of subscriptions whose top-k results contain o' from the result index that maintains the top-k results of all the subscriptions, and we update the object index. For each subscription in the set, we evaluate the new top-k result and its safe region by using the object index. The new top-k result and the safe region are sent to the user, and the subscription and the result indices are also updated.

Movement of user. Recall that users receive and maintain their safe regions at the same time as their top-k results. Therefore, they can check whether they exit their safe regions or not by

TABLE I: Summary of notations

Notation	Definition
o	Spatio-textual object
O	Set of objects generated and not expired so far
s	MkSK subscription
S	Set of MkSK subscriptions
w	Keyword
t_{max}	Maximum size of $\|s.t\|$
R	Safe region
c	Centroid of R
$\lambda(s, o\|\cdot)$	Spatial proximity threshold for s
n	Node of the SQ-tree
S_n^r	Set of subscriptions in the subtree rooted at n
S_n	Set of subscriptions maintained in n
T_n^r	Set of keywords contained in $s \in S_n^r$
I_n	Inverted file
$\Lambda_n[\cdot], \Lambda_n^r[\cdot]$	Spatial proximity threshold for S_n, S_n^r

themselves. When a user u exits his/her safe region, u reports the current location to Lamps. Lamps deals with this case as with the case of object expirations.

In the following section, we introduce the details of how to process object generations and expirations. (Recall that Lamps processes those cases in which users exit their safe regions in the same way as object expirations.) Table I summarizes the mathematical notations used throughout this paper.

IV. Process of object generation

When a new object o is generated, we need to identify the set of subscriptions whose top-k results change by o. An efficient approach achieving this is to maintain subscriptions by using a spatial index, such as R-tree and Quad-tree. However, because the current locations of subscriptions are unknown, we cannot simply maintain them by using a spatial index. To solve this problem, Lamps employs a novel index, namely SQ-tree. This index maintains MkST subscriptions based on their safe regions and a Quad-tree, and it can filter a set of subscriptions whose top-k results do not change by o without requesting their current locations. In this section, we first propose a novel filtering technique in Section IV-A. Then we present the detailed SQ-tree in Section IV-B. We introduce the algorithm for object generation in Section IV-C and how to update SQ-tree is presented in Section IV-D.

A. Filtering technique

Given a newly generated object o, we need to identify all subscriptions whose top-k results change by o. We denote the k-th smallest score of a subscription s as $kscore(s)$. Then the top-k result of s needs to be updated if $kscore(s) > score(s, o)$. A lot of works have studied spatio-textual search, and it is well known that text-first filtering techniques have good performances [3]. In this section, we propose a novel filtering technique, which filters a set of subscriptions whose top-k results do not change, based on the number of common keywords between $s.t$ and $o.t$.

Assume that $s.p$ is known. (This assumption is removed later.) Let i be the number common keywords between $s.t$ and $o.t$, i.e, $|s.t \cap o.t| = i$. Then we can compute a lower bound of $text(s.t, o.t)$, $text_{lb}(s.t, o.t|i)$, for s such that $|s.t \cap o.t| = i$:

LEMMA 3. $text_{lb}(s.t, o.t|i) = 1 - \frac{i}{|s.t|}$.

PROOF. If $|o.t| = i$, $text(s.t, o.t)$ takes the smallest value. From $text(s.t, o.t) = 1 - \frac{i}{|s.t|+|o.t|-i}$, the lemma holds. □

Based on Lemma 3 and Equation 1, we can derive a spatial proximity threshold, $\lambda(s, o|i)$, for s such that $|s.t \cap o.t| = i$:

$$\lambda(s, o|i) = \frac{kscore(s)}{\alpha} - \frac{1-\alpha}{\alpha} \cdot text_{lb}(s.t, o.t|i). \quad (5)$$

The following theorem is immediately derived from Equation 5.

THEOREM 3. *A newly generated object o does not become the top-k result of s if $\lambda(s, o|i) < dist(s.p, o.p)$.*

PROOF. The top-k result of s needs to be updated if $kscore(s) > score(s, o)$. If $\lambda(s, o|i) < dist(s.p, o.p)$, $kscore(s) < \alpha \cdot dist(s.p, o.p) + (1-\alpha) \cdot text_{lb}(s.t, o.t|i)$. Because $text_{lb}(s.t, o.t|i) \leq text(s.t, o.t)$, $\alpha \cdot dist(s.p, o.p) + (1-\alpha) \cdot text_{lb}(s.t, o.t|i) \leq \alpha \cdot dist(s.p, o.p) + (1-\alpha) \cdot text(s.t, o.t) = score(s, o)$. Then the theorem holds. □

Theorem 3 claims that we can safely prune s if $\lambda(s, o|i) < dist(s.p, o.p)$.

We have assumed that $s.p$ is known. However, without requesting the current locations of all users, the above filtering does not function, because $kscore(s)$ is determined by $s.p$ and $\lambda(s, o|i)$ is also determined by $s.p$. To safely prune s regardless of where s is in its safe region R, we modify the above filtering technique. Let $kscore_{ub}(s)$ be the upper bound of $kscore(s)$ when $s.p$ is any position in R. We can compute $kscore_{ub}(s)$:

LEMMA 4. $kscore_{ub}(s) = \frac{\gamma \cdot \alpha + kscore(s)}{2}$

PROOF. Recall that R is the intersection of $\mathcal{E}_{o^*}(o^* \in A)$ and $\mathcal{E}_{o^*}$ is an ellipse with $s.p$ and $o^*.p$ as two foci. Given $\mathcal{E}_{o^*}$ and the straight line passing through two points $s.p$ and $o^*.p$, we have two intersection points between $\mathcal{E}_{o^*}$ and the line. Let p' be the intersection point farther from o^* than the other intersection point. When $s.p = p'$, $score(s, o^*)$ takes the largest value. Therefore, the upper bound of $score(s, o^*)$, $score_{ub}(s, o^*)$ is calculated as follows:

$$\begin{aligned} score_{ub}(s, o^*) &= \alpha \cdot dist(p', o^*.p) + (1-\alpha) \cdot text(s.t, o^*.t) \\ &= \alpha \cdot (dist(p', o^*.p) + r_{o^*}). \end{aligned}$$

Besides, $dist(s.p, p') = dist(p', o^*.p) - dist(s.p, o^*.p)$, so $dist(p', o^*.p) = \frac{\gamma - r_{o^*} + dist(s.p, o^*.p)}{2}$ from Equation 4. Then,

$$\begin{aligned} score_{ub}(s, o^*) &= \alpha \cdot (\frac{\gamma - r_{o^*} + dist(s.p, o^*.p)}{2} + r_{o^*}) \\ &= \frac{\gamma \cdot \alpha + score(s, o^*)}{2}. \end{aligned}$$

Because $kscore_{ub}(s) = \max\{score_{ub}(s, o^*) | o^* \in A\}$, the lemma holds. □

Based on Equation 5 and Lemma 4, we can derive the spatial proximity threshold when $s.p$ is any position in R:

$$\lambda(s, o|i) = \frac{kscore_{ub}(s)}{\alpha} - \frac{1-\alpha}{\alpha} \cdot text_{lb}(s.t, o.t|i). \quad (6)$$

The following theorem is derived from Equation 6.

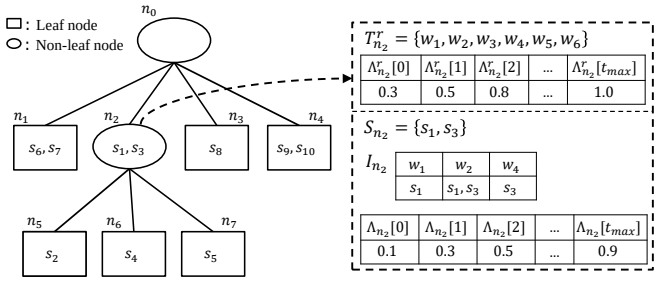


Fig. 6: Illustration of an SQ-tree

THEOREM 4. *A newly generated object o does not become the top-k result of s if $\lambda(s, o|i) < dist(s.R, o.p)$, where $dist(s.R, o.p)$ is the minimum value of the spatial proximity between any position in $s.R$ and $o.p$.*

PROOF. In the same way as Theorem 3. □

Theorem 4 claims that we can safely prune s and do not need to update $s.R$ if $\lambda(s, o|i) < dist(s.R, o.p)$.

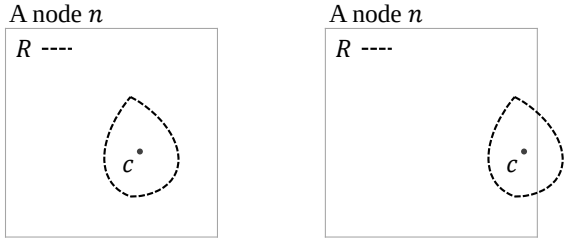
B. SQ-tree

SQ-tree maintains MkST subscriptions based on their safe regions and a Quad-tree as shown in Fig. 6, and *by exploiting the filtering technique introduced is Section IV-A, it can filter a set of subscriptions whose top-k results do not change by o without requesting their current locations.* Each node of an SQ-tree maintains the subscriptions whose centroids of safe regions are in the area of the node. Moreover, unlike conventional spatial indices, the SQ-tree maintains subscriptions not only in leaf nodes but also in non-leaf nodes. Let S_n be a set of subscriptions maintained in a node n and S^r_n be a set of subscriptions maintained in the subtree rooted at n. A node n stores a set of keywords $T^r_n = \cup_{s \in S^r_n} s.t$ and an inverted file I_n for the subscriptions contained in S_n.

In addition, n stores the spatial proximity thresholds $\Lambda_n[i]$ and $\Lambda^r_n[i]$ for each $i \in [0, t_{max}]$ to filter a set of subscriptions whose top-k results do not change by o. Here, let $dist(n, o.p)$ be the minimum value of the spatial proximity between $o.p$ and n, and $dist(n, o.p) = 0$ if $o.p$ is in the area of n (denoted by $o.p \in n$). We filter a set of subscriptions by using these thresholds based on $dist(n, o.p) \leq dist(s.R, o.p)$. When a subscription s is maintained in n, however, there are two cases: R is fully contained in n (denoted by $R \subseteq n$) and R is partially contained in n (denoted by $R \nsubseteq n$). Let c be the centroid of R. In the right case in Fig. 7, i.e., $R \nsubseteq n$, $dist(n, o.p) \leq dist(R, o.p)$ does not always hold. To solve this problem, we consider an additional spatial proximity, $add(R, n)$, which indicates how far R protrudes from n. It is calculated as follows:

$$add(R, n) = \begin{cases} 0 & (R \subseteq n) \\ dist_f(c, R) - dist_n(c, n) & (R \nsubseteq n) \end{cases} \quad (7)$$

where $dist_f(c, R)$ is the spatial proximity from c to the farthest boundary of R and $dist_n(c, n)$ is the spatial proximity from



(a) $R \subseteq n$ (b) $R \nsubseteq n$

Fig. 7: Example of maintaining s in n

c to the nearest boundary of n. Then the following inequality holds:

$$dist(n, o.p) \leq dist(R, o.p) + add(R, n) \quad (8)$$

Then we define $\Lambda^r_n[i]$ and $\Lambda_n[i]$ as follows, respectively:

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

where $\lambda(s, o|i) = \lambda(s, o||s.t|)$ if $|s.t| < i$.

EXAMPLE 6. *Assume that subscriptions exist as shown in Fig. 2 and that each triangle indicates the centroid of a subscription. Then an SQ-tree is constructed as shown in Fig. 6. For example, $S^r_{n_2} = \{s_1, s_2, s_3, s_4, s_5\}$, so a set of their keywords and $\Lambda^r_n[\cdot]$ are stored in n_2. Because $S_n = \{s_1, s_3\}$, n_2 stores an inverted file of their keywords and $\Lambda_n[\cdot]$.*

SQ-tree construction. Since the scoring function is based on the spatial proximity and the textual relevance, it is desirable that a subscription index maintains subscriptions based on both the factors. However, a spatial index, such as R-tree and Quad-tree, focuses only on the spatial factor during index construction, regardless of the textual factor. To solve this problem, an SQ-tree maintains subscriptions not only in leaf nodes but also in non-leaf nodes based on the textual factor. This is because, if a non-leaf node n maintains subscriptions, for each child node n' of n, the number of subscriptions maintained in the subtree rooted at n', i.e. $|S^r_{n'}|$, decreases. Thereby, each spatial proximity threshold $\Lambda^r_n[\cdot]$ of n' becomes tight, and n' tends to be pruned.

We then introduce an algorithm that determines the node for maintaining subscriptions. We consider whether a subscription s should be maintained in a node n. Let $E[dist(n, o.p)]^2$ be an expected value of $dist(n, o.p)$ when an object o is generated. Then the probable number of common keywords, $pnum(s, n)$, where the top-k result of s needs to be updated, is calculated as follows:

$$pnum(s, n) = \lfloor (1 - \frac{score_{ub}(s) - \alpha \cdot E[dist(n)]}{1 - \alpha}) \cdot |s.t| \rfloor. \quad (9)$$

Moreover, we assume that the distribution of object keywords does not change much over time. Then we define the expected value of $i_n = |T^r_n \cap o.t|$, $E[i_n]$, as follows:

$$E[i_n] = \sum_{w \in T^r_n} \frac{|O_w|}{|O|} \quad (10)$$

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

Algorithm 1: Insert(n, s)

```
   Input: n and s
1  S_n^r ← S_n^r ∪ {s}
2  Update T_n^r, E[i_n], and, Λ_n^r[·]
3  if n is a leaf node then
4      S_n ← S_n ∪ {s}
5      Update I_n and Λ_n[·]
6      if |S_n| is larger than the node capacity then
7          Create child nodes, distribute subscriptions to child
             nodes, and recompute Λ_n[·].
8  else
9      Distribute subscriptions to child nodes.
10     if E[i_n] < pnum(s, n) then
11         S_n ← S_n ∪ {s}
12         Update I_n and Λ_n[·]
13     else
14         for ∀n' ∈ N //N is n's children do
15             if s.c ∈ n' then
16                 Insert(n', s)
```

where O_w is a set of objects whose t contains a keyword w and $E[i_n] = t_{max}$ if $E[i_n] \geq t_{max}$. When an object is generated, s tends to be pruned if $E[i_n] < pnum(s, n)$. Thus subscriptions that satisfy this condition are maintained in n even if n is not a leaf node. In Section VII-C, we verify the accuracy of $E[i_n]$.

Algorithm 1 describes how to insert a subscription s into a node n of the SQ-tree[3]. We first update S_n^r, T_n^r, $E[i_n]$, and $\Lambda_n^r[\cdot]$ (lines 1–2). If n is a leaf node, we update S_n, I_n, and $\Lambda_n[\cdot]$ (lines 3–7). Furthermore, if $|S_n|$ is larger than the node capacity, which is the maximum number of subscriptions that a leaf node can maintain, we create child nodes and distribute subscriptions with $pnum(s, n) \geq E[i_n]$ to the child nodes. After that, we recompute $\Lambda_n[\cdot]$ upon subscriptions that are not distributed (lines 6–7). On the other hand, if n is not a leaf node, we distribute subscriptions with $pnum(s, n) \geq E[i_n]$ to its child nodes (line 9). If $E[i_n] < pnum(s, n)$, we update S_n, I_n, and $\Lambda_n[\cdot]$. If $E[i_n] \geq pnum(s, n)$, we recursively execute Insert(n', s) where n' is the child node of n if c is contained in n' (lines 10–16).

SQ-tree filtering. We introduce how to filter the subscriptions maintained in each node of the SQ-tree. This filtering is performed in two steps (node filtering and subscriptions filtering). We assume that an object o is generated and we check a node n.

Subscriptions filtering. Here, we focus on how to filter subscriptions maintained by a node n. Let i_{min} be the smallest i such that $\Lambda_n[i] \geq dist(n, o.p)$. i_{min} indicates the minimum required number of common keywords for o to become the top-k result of each subscription in S_n. So, we have the following theorem:

THEOREM 5. *Given a newly generated object o and i_{min}, o does not become the top-k result of $s \in S_n$ if $|s.t \cap o.t| < i_{min}$.*

[3]When we construct the SQ-tree from scratch, we regard the spatial area $\mathbb{R}^2$ as the root node of the SQ-tree, n_{root}. The SQ-tree can be constructed by executing Insert(n_{root}, s) for each subscription $s \in S$.

PROOF. If $|s.t \cap o.t| = j < i_{min}$, $\Lambda_n[j] < dist(n, o.p)$. Because $\lambda(s, o|j) + dist_{add}(s.R, n) \leq \Lambda_n[j]$, $\lambda(s, o|j) + dist_{add}(s.R, n) < dist(n, o.p)$. From Equation 8, $\lambda(s, o|j) + dist_{add}(s.R, n) < dist(s.R, o.p) + dist_{add}(s.R, n)$. Therefore, $\lambda(s, o|j) < dist(s.R, o.p)$. Then, from Theorem 4, the theorem holds. □

From Theorem 5, we can safely prune subscriptions s such that $|s.t \cap o.t| < i_{min}$, and we do not need to update their top-k results and safe regions.

EXAMPLE 7. *In Figs. 2 and 6, we assume that o_9 is generated and we check n_2. Because $\Lambda_{n_2}[1] < dist(n_2, o_9.p) < \Lambda_{n_2}[2]$, $i_{min} = 2$. Therefore, the top-k result of each subscription $s \in S_n$, i.e., s_1 and s_3, may be updated by o_9 if $|s.t \cap o.t| \geq 2$. From Fig. 2, $|s_1.t \cap o_9.t| = 2$ and $|s_3.t \cap o_9.t| = 1$. We therefore add only s_1 to the set of candidate subscriptions.*

Node filtering. We next present how to filter all subscriptions contained in a subtree rooted at n. Recall that T_n^r is a set of keywords contained in $s \in S_n^r$. Given $i_n = \min\{|T_n^r \cap o.t|, t_{max}\}$, we have the following theorem:

THEOREM 6. *If $\Lambda_n^r[i_n] < dist(n, o.p)$, a newly generated object o does not become the top-k result of $s \in S_n^r$.*

PROOF. If $|s.t \cap o.t| = j$ for each subscription $s \in S_n^r$, we have $j \leq i_n$. Then $\lambda(s, o|j) \leq \lambda(s, o|i_n)$, because $\lambda(s, o|i)$ monotonically increases as i increases. If $\Lambda_n^r[i_n] < dist(n, o.p)$, $\lambda(s, o|i_n) < dist(s.R, o.p)$, which can be proved in the same way as Theorem 5. Therefore, $\lambda(s, o|j) < dist(s.R, o.p)$. Then, from Theorem 4, Theorem 6 holds. □

Theorem 6 claims that we can safely prune n, i.e., we do not need to update the top-k results and safe regions of all subscriptions contained in S_n^r if $\Lambda_n^r[i_n] < dist(n, o.p)$.

EXAMPLE 8. *In Figs. 2 and 6, we assume that o_{10} is generated and we check n_2. On the other hand, when o_{10} is generated, the common keyword between $T_{n_2}^r$ and $o_{10}.t$ is w_2, so $i_{n_2} = 1$. Since $dist(n_2, o_{10}.p) = 0.6$, $\Lambda_{n_2}^r[i_n] < dist(n_2, o_{10}.p)$. Therefore, n_2 is pruned. That is, the top-k results and the safe regions of all subscriptions contained in $S_{n_2}^r$ are not updated by o_{10}.*

C. Algorithm for object generation

Algorithm 2 describes the SQ-tree based algorithm for object generation. Following the filtering-and-verification paradigm, we first obtain a set of candidate subscriptions by using SQ-tree (lines 1–15), and then verify the candidate subscriptions (line 16). Specifically, we first initialize a set S_g of candidate subscriptions and a heap H to keep the nodes of SQ-tree which have not been pruned, and the root node of SQ-tree is pushed H (lines 1–2). Each node n popped in line 4 is processed as follows. We first calculate i_n (line 5). Next, If n is not pruned by node filtering (line 6) and n is not a leaf node, each child node n' of n such that $S_{n'}^r \neq \varnothing$ is pushed into H (lines 7–10). If $S_n \neq \varnothing$, in addition, we execute the subscriptions filtering for subscriptions contained in S_n (lines 12–15). We first obtain i_{min} (line 12). If $i_n < i_{min}$, we have $|s.t \cap o.t| < i_{min}$ for each subscription $s \in S_n$. In this case,

Algorithm 2: ObjectGeneration(o)

```
   Input: o
1  S_g ← ∅, H ← ∅ //S_g is a set of candidate subscriptions
2  Push root node of SQ-tree into H
3  while H ≠ ∅ do
4      n ← the node popped from H
5      i_n ← min(|T_n^r ∩ o.t|, t_max)
6      if Λ_n^r[i_n] ≥ dist(n, o.p) then
7          if n is not a leaf node then
8              for ∀n' ∈ N //N is n's children do
9                  if S_n'^r ≠ ∅ then
10                     Push n' into H
11         if S_n ≠ ∅ then
12             i_min ← min{i|Λ_n[i] ≥ dist(n, o.p)}
13             if i_min ≤ i_n then
14                 for ∀s ∈ S_n where |s.t ∩ o.t| ≥ i_min do
15                     S_g ← S_g ∪ {s}
16 UpdateResult(S_g)
```

all subscriptions contained in S_n are pruned. Therefore, iff $i_{min} \leq i_n$, we add $s \in S_n$, where $|s.t \cap o.t| \geq i_{min}$, to S_g (lines 13–15). After filtering, we execute UpdateResult(S_g) for S_g (line 16). For each subscription $s \in S_g$, we need to update the top-k result and the safe region of s. Specifically, we ask the user who issued s for his/her current location, update the scores of the current top-k results, and calculate $score(s, o)$. If $score(s, o) < kscore(s)$, the top-k result of s is updated (i.e., o becomes a new top-k result of s) and the object whose score for s was the k-th smallest before the top-k result update becomes o_{k+1}. On the other hand, if $score(s, o) \geq kscore(s)$, the top-k result of s does not change. Because we need to update the safe region of s and o is not necessarily o_{k+1}, we find o_{k+1} by using the object index. (We introduce the details later.) Then we compute a new safe region of s by using the ASR construction algorithm.

D. SQ-tree maintenance

When the top-k result and the safe region of a subscription s are updated, we need to update the SQ-tree, because the node that maintains s and the information stored in each node may change. If the centroid of the updated safe region of s is contained in n where n maintains s, we update the information stored in n and the ancestor nodes of n if necessary. Otherwise, we remove s from the SQ-tree once and insert it again based on the updated safe region.

V. Process of object expiration

When an object o' expires, we need to re-evaluate the top-k results and the safe regions of subscriptions whose top-k results contain o'. To obtain such subscriptions, we use the result index that maintains objects that are top-k results of at least one subscription. When we re-evaluate the top-k result and the safe region of a subscription s, we have to retrieve the objects whose scores for s are the k-th and $(k+1)$-th smallest, o_k and o_{k+1}. To retrieve such objects efficiently, Lamps employs the object index, namely IQF, which is composed of **I**nverted file, and **Q**uad-tree for each **F**requent keyword. Recall that an object must contain at least one common keyword with a subscription s to become the top-k result of s. Therefore, IQF maintains objects with an inverted file. If a keyword $w \in s.t$ is contained in a lot of objects, however, it is time-consuming to check all the objects contained in the posting list of w, $PL[w]$. To solve this problem, we employ the concept of Inverted linear Quad-tree, which creates a Quad-tree for each keyword and is a state-of-the-art index for spatio-textual top-k search [25]. If we create Quad-trees for all keywords, however, we incur a large memory cost. Therefore, we create a Quad-tree, Q_w, for each frequent keyword w. In this paper, frequent keywords are defined as the keywords whose appearance frequencies are in the top-x percent among all keywords. (If x is large, we create $Q_{w'}$ for a non-frequent keyword w', and it incurs a large memory cost. Therefore, we determine x empirically.)

Here, we focus on how to retrieve o_k. Recall that we need o_{k+1} to calculate the ASR. We obtain it at the same time as we retrieve o_k. For each object $o \in O_{s.t} \backslash s.A$, if $|s.t \cap o.t| = i$, the lower bound of $score(s, o)$, $score_{lb}(s, o, |i)$, is calculated as follows:

$$score_{lb}(s, o|i) = (1 - \alpha) \cdot \frac{i}{|s.t|}. \tag{11}$$

Let o_b be the object that has the best score among the objects checked so far. Then we have the following theorem:

THEOREM 7. *If $score_{lb}(s, o|i) \geq score(s, o_b)$, each object $o \in O \backslash s.A$ such that $|s.t \cap o.t| \leq i$ does not become o_k.*

PROOF. For each object $o \in O_{s.t} \backslash s.A$ such that $|s.t \cap o.t| \leq i$, $dist(s.p, o.p) \geq 0$ and $text(s.t, o.t) \geq 1 - \frac{i}{|s.t|}$. Therefore, $score(s, o) > score(s, o_b)$. Then the theorem holds. □

From Theorem 7, we can safely prune objects such that $|s.t \cap o.t| < i$ if $score_{lb}(s, o|i) \geq score(s, o_b)$.

Algorithm 3 describes how to retrieve o_k. Because the object with a large $|s.t \cap o.t|$ is likely to become o_k, we should check each object o in descending order of $|s.t \cap o.t| = i$. If $i > 1$, for each object $o \in O_{s.t} \backslash s.A$ with $|s.t \cap o.t| = i$, we set o to o_k if $score(s, o) < score(s, o_k)$. (lines 7–9). On the other hand, if $i = 1$, for each keyword $w \in s.t$, we execute Search-QF(Q_w, s, o_k), where w is a frequent keyword. Otherwise, we set o to o_k for each object o contained in $PL[w]$ if $score(s, o) < score(s, o_k)$. (lines 10–16). Search-QF(Q_w, s, o_k) checks the objects in Q_w closer to $s.p$ in order and sets o to o_k if $o \notin s.A$ and $score(s, o) < score(s, o_k)$. If the scores of unchecked objects are never smaller than $score(s, o_k)$, we terminate the retrieval.

EXAMPLE 9. *Assume that the objects exist as shown in Fig. 2 and w_1 and w_2 are frequent keywords. Then IQF is constructed as shown in Fig. 8. Moreover, we assume that o_1 expires, and we re-calculate the top-k result of s_2. Since $s_2.t = \{w_1, w_4\}$, we first check the objects whose keywords contain both w_1 and w_4, i.e., o_3 and o_5. If $score_{lb}(s_2, o|1) < score(s_2, o_k)$ after checking those objects, we need to check the objects whose keywords contain w_1 or w_4. When we check the objects whose*

Algorithm 3: RetrieveObject(s)

```
Input: s, O
o_k ← ∅
for ∀o ∈ O_{s.t}\s.A do
    Compute |s.t ∩ o.t| by using Inverted file
i ← |s.t|
while i > 0 ∧ score_lb(s, o, |i) < score(s, o_k) do
    if i > 1 then
        for ∀o ∈ O_{s.t}\s.A such that |s.t ∩ o.t| = i do
            if score(s, o) < score(s, o_k) then
                o_k ← o
    else
        for ∀w ∈ s.t do
            if w is a frequent keyword then
                Search-QF(Q_w, s, o_k)
            else
                for ∀o ∈ PL[w]\s.A do
                    if score(s, o) < score(s, o_k) then
                        o_k ← o
    i ← i − 1
return o_k
```

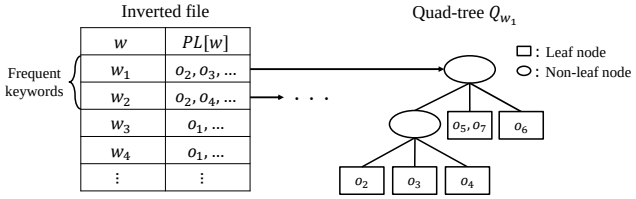


Fig. 8: Example of IQF

keywords contain w_1, we check the objects in Q_{w_1} closer to $s_2.p$ in order. On the other hand, when we check the objects whose keywords contain w_4, we check all objects in $PL[w_4]$. Finally, the object, whose score is the smallest among the checked objects, becomes the new top-k result of s_2.

VI. Discussion

Textual similarity functions. In this paper, we use Jaccard similarity for textual similarity function. If we use Dice or Cosine similarities, it is enough to change Lemma 3 based on their definitions. In our experiment, we verify the performance when we use each similarity function.

Procedure of subscription issue and delete. In this paper, we assume that subscriptions are pre-registered in our system and the number of subscriptions does not change. If a new subscription s is issued, we can compute the top-k result of s in the same way as in Algorithm 3, and we then retrieve $(k+1)$ objects whose scores for s are the smallest. After that, we compute the ASR of s and its centroid and then insert s into the SQ-tree. If a subscription s' is deleted, we remove s' from the SQ-tree and update information stored in the node n, which maintain s', and the ancestor nodes of n if necessary.

VII. Experiment

In this section, we conduct extensive experiments to verify the efficiency of Lamps. All algorithms were implemented in C++, and all experiments were conducted on a PC with 3.20GHz Intel Core i7 processor and 64GB RAM. Following the typical setting of pub/sub systems (e.g., [20], [21]), we assume that the indices fit in the main memory to support real-time response.

A. Experimental Setup

Since this is the first work of the problem of monitoring top-k spatio-textual objects for a large number of moving top-k spatio-textual subscriptions, there is no existing method. Therefore, we compared Lamps with Naive and three methods that adopt some of the functions of Lamps: Lamps w/o ASR, Lamps w/o SQ-tree, Lamps w/o IQF. In Lamps w/o ASR, we use LSR instead of ASR for the safe region of each subscription. Because there is no existing index maintaining moving top-k subscriptions, Lamps w/o SQ-tree accesses all subscriptions and update the results when an object is generated. Lamps w/o IQF adopts IQ-tree instead of IQF to maintain objects. IQ-tree is a hybrid structure of inverted file and Quad-tree, like IR-tree [30].

Object datasets. We used two real datasets to simulate the object stream. *TWEETS* consists of 20 million geo-tagged tweets located inside the United States [2]. *PLACES* consists of 9.4 million public places inside the United States [14]. Each entry in *PLACES* includes the geo-location and the set of keywords. The statistics of the two datasets are summarized in Table II.

Trajectory datasets. We used real and synthetic trajectories to simulate the movements of users. For the real trajectories, we used the GPS records from *OSM*[4]. In our experiments, timestamp (ts), was used to capture the periodicity of location update. We set each ts to be 1 second in the experiments. We extracted 1,000,000 trajectories, and each trajectory contains 100 sequential points. For the synthetic trajectories, we generated 1,000,000 trajectories based on the intersections and the roads of the United States[5] (*SUS*). Each trajectory contains 1,000 sequential points.

Subscription workload. We generated $|S|$ MkST subscriptions based on the object datasets. (The default $|S|$ is 250,000.) For each subscription, we randomly picked l keywords from a tweet or an entry as subscription keywords $(1 \leq l \leq 5)$. For the trajectory of each subscription, we randomly chose a trajectory from the extracted or generated trajectories. For each subscription, in addition, the preference parameter α was randomly chosen from (0, 1) and k was a random number between 1 and k_{max}. Throughout the experiments, we set k_{max} to 10, unless otherwise specified.

Object workload. Our experiment starts after $|O_{init}|$ initial objects are generated. (The default $|O_{init}|$ is 1,000,000.) For

[4]https://planet.openstreetmap.org/gps/

[5]https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm

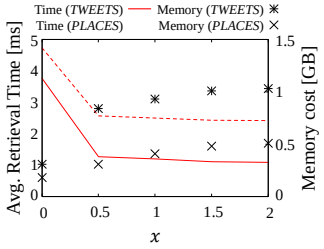


Fig. 9: Impacts of x

TABLE III: Average update time [ms] of each method

Algorithm	*TWEETS*		*PLACES*	
	OSM	*SUS*	*OSM*	*SUS*
Naive	6657.9	5858.8	11242.1	9666.6
Lamps w/o ASR	2124.8	1746.1	2616.7	3399.3
Lamps w/o SQ-tree	2436.6	2424.7	2366.2	2320.5
Lamps w/o IQF	155.3	158.0	215.1	210.1
Lamps	80.0	75.2	156.3	141.2

TABLE IV: Memory cost [GB] of each method

method	*TWEETS*	*PLACES*
Lamps w/o SQ-tree	0.84	0.31
Lamps w/o IQF	1.63	0.57
Lamps	1.03	0.42

TABLE II: Dataset statistics

Dataset	**TWEETS**	**PLACES**
# of objects	20M	9.4M
Vocabulary size	2.1M	54K
Avg. # of object keywords	5.2	2.9

each timestamp, we generate f object updates including object generations and object expirations. (The default f is 100.) Let ϵ be the percentage of expirations, object updates include $f * (1 - \epsilon)$ object generations and $f * \epsilon$ object expirations. Note that object expirations in the real applications introduced in Section I are very rare compared with object generations [18]. Therefore, ϵ is a small value and we used 0.1 as the default value of ϵ.

We report the update time (the average time to update the top-k results of subscriptions against object generations/expirations and movements of users per ts). The update time includes the generated objects' processing (GOP) time, the expired objects' processing (EOP) time, and the movements of users processing time.

B. Experimental tuning

First, we tune the value of x, where x the percentage of frequent keywords among all keywords. Recall that we create Q_w for each keyword w whose appearance frequency is in the top-x percent among all keywords (See Section V).

Fig. 9 reports the retrieval time (the average time to retrieve o_{k+1}) and the memory cost against the two datasets. Although the memory cost increases with the increase of x, the retrieval time is almost constant when $x \geq 0.5$. Therefore, we set 0.5 as the default x value in the following experiments, because it strikes a good trade-off between the retrieval time and the memory cost.

C. Experimental results

Comparison between LSR and ASR. Fig. 10 reports the computation times (the average times required to compute the safe regions of users who exit their safe regions per ts) of the LSR and the ASR using the two datasets where each parameter follows the default setting. The number of subscriptions that exit their safe regions per ts in ASR case is slightly larger than that in LSR case, because ASR tends to be smaller than LSR. However, the ASR is computed very fast by avoiding computation of the dominant regions and their intersections, so, at a timestamp, ASR computation does not become an overhead of result update even if the number of subscriptions that exit their safe regions is large.

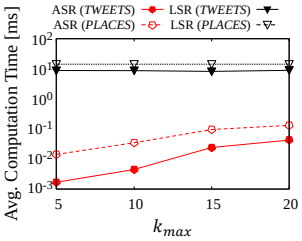


Fig. 10: Comparison between LSR and ASR

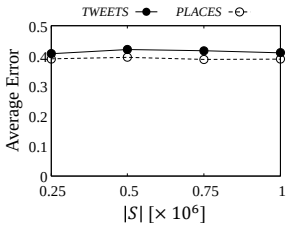


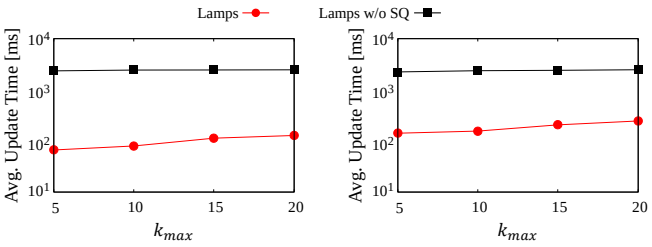
Fig. 11: Accuracy of $E[i_n]$

Accuracy of $E[i_n]$. We verify the accuracy of $E[i_n]$ in Fig. 11 by using each datasets and varying the number of subscriptions $|S|$. Fig. 11 reports the average error (the average of $|E[i_n] - i_n|$ calculated in the node filtering). We see that $E[i_n]$ is sufficiently accurate, because the average error is less than 1 regardless of $|S|$.

Overall performance. Table III shows the update time of each method where each parameter is the default setting. Lamps is the fastest among all the methods, regardless of the object and trajectory datasets. When we use *TWEETS*, the update time is longer than when we use *PLACES*. This is because the vocabulary size of *PLACES* is smaller than that of *TWEETS*. Each object and subscription tend to have the same keywords, thereby the number of subscriptions whose top-k results may change by a newly generated object increases and the posting list of each keyword is large. As mentioned above, ASR has much better performance than LSR. Therefore, Lamps is much faster than Lamps w/o ASR. Moreover, Lamps is much faster than Lamps w/o SQ-tree because Lamps can filter subscriptions whose top-k results do not change by using SQ-tree when an object is generated.

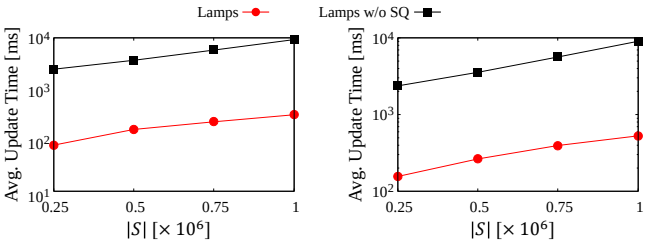
Although Lamps is faster than Lamps w/o IQF, Lamps has an advantage in the aspect of memory cost. Table IV shows the memory cost of each method when we use *OSM*. Lamps needs less memory cost than Lamps w/o IQF. Recall that Lamps w/o IQF adopts IQ-tree instead of IQF to maintain objects. IQ-tree incurs a large memory cost, because each node of IQ-tree creates an inverted file for keywords of objects maintained in the subtree rooted at the node.

In the following experiments, we compared Lamps with Lamps w/o SQ-tree. When we examined the impact of frequency of object updates, we add Lamps w/o IQF as a competitor. Moreover, we use *OSM* for trajectory dataset since the update time of each dataset is almost the same.



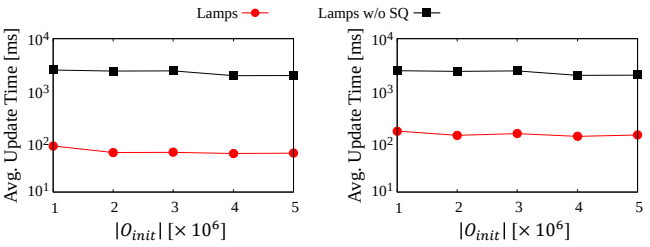
(a) *TWEETS* (b) *PLACES*

Fig. 12: Impact of k_{max}



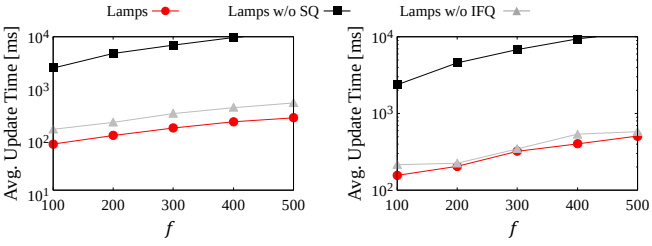
(a) *TWEETS* (b) *PLACES*

Fig. 13: Impact of $|S|$



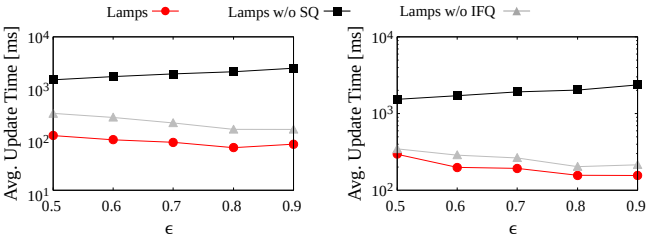
(a) *TWEETS* (b) *PLACES*

Fig. 14: Impact of $|O_{init}|$



(a) *TWEETS* (b) *PLACES*

Fig. 15: Impacts of f



(a) *TWEETS* (b) *PLACES*

Fig. 16: Impacts of ϵ

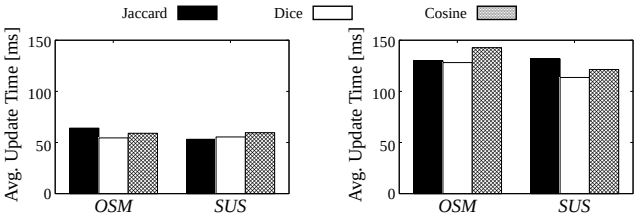
Impact of k_{max}. We examined the impact of k_{max} and Fig. 12 shows the result. Lamps consistently updates the top-k results faster than Lamps w/o SQ. Both methods need a longer update time as k_{max} increases. When k_{max} increases, the possibility that an expired object is contained in the top-k results for the subscriptions becomes high. Because the number of recalculations of the top-k results and safe regions increases, the EOP time increases. When k_{max} increases, moreover, the average $kscore(s)$ of the subscriptions becomes larger, so a newly generated object is more likely to affect more top-k results of the subscriptions. With a large k_{max}, the GOP time of Lamps increases. On the other hand, the GOP time of Lamps w/o SQ-tree is constant because Lamps w/o SQ-tree always checks the top-k results of all subscriptions when an object is generated. Therefore, the difference in update time between Lamps and Lamps w/o SQ-tree becomes slightly smaller as k_{max} increases.

Impact of number of subscriptions. We examined the impact of number of subscriptions, $|S|$, and Fig. 13 shows the result. Lamps updates the top-k results faster than Lamps w/o SQ. Both methods need a longer update time as $|S|$ increases. The difference in update time between Lamps and Lamps w/o SQ-tree becomes larger as $|S|$ increases (Note that Fig. 13 plots the result on log scale.). Lamps w/o SQ-tree checks the top-k results of all subscriptions when an object is generated. On the other hand, Lamps can filter subscriptions whose top-k results do not change. Because the difference in the number of subscriptions whose top-k results are updated becomes larger as $|S|$ increases, the difference in update time becomes larger.

Impact of number of initial objects. We examined the impact of number of initial objects, $|O_{init}|$, and Fig. 14 shows the result. Lamps updates the top-k results faster than Lamps w/o SQ. Both methods need a slightly faster update time as $|O_{init}|$ increases. When $|O_{init}|$ increases, the number of users that exit their safe regions and the number of objects checked when the top-k results and the safe regions are recalculated increases. However, a newly generated object will affect fewer top-k results of subscriptions, and the possibility that an expired object is contained in the top-k results of the subscriptions becomes low. Therefore, the update time slightly decreases.

Impact of frequency of object updates. We examined the impact of frequency of object updates, f, Fig. 15 shows the result, and Lamps is a clear winner. Moreover, all methods need a longer update time as f increases. This is because all methods process newly generated and expired objects one by one, so the GOP and EOP time of all methods increases linearly.

Impact of percentage of expirations. We examined the impact of percentage of expirations, ϵ. Recall that f object updates include $f * (1 - \epsilon)$ object generations and $f * \epsilon$ object expirations. Fig. 16 shows the result and Lamps is a clear winner. Moreover, all methods need a longer update time as f increases. This is because all methods process newly generated and expired objects one by one, so the GOP and EOP time of all methods increases linearly.



(a) *TWEETS* (b) *PLACES*

Fig. 17: Impact of textual similarity functions

Impact of textual similarity functions. We examined the impact of textual similarity functions; Figure 17 shows the result of Lamps when we used each function. As shown in Figure 17, the update time is almost the same for all functions.

VIII. Related Work

In this section, we review the existing studies of spatio-textual top-k search, pub/sub systems, and moving query processing.

Spatio-textual top-k query. Spatio-textual top-k query has been widely studied [6], [17], [25], [26]. These studies proposed methods that combine both a spatial index (e.g., R-tree and Quad-tree) and a textual index (e.g., an inverted file). A good summary of spatio-textual query processing is available in [3], and text-first filtering techniques have good performances. In addition, [25] proposed inverted linear Quad-tree (IL-Quad-tree), which creates a Quad-tree for each keyword. IL-Quad-tree is a state-of-the-art index for spatio-textual top-k query, thus we employ the concept of IL-Quad-tree for the object index of Lamps. Recently, a variety of spatio-textual top-k queries have been studied, for example, reverse spatio-textual query [29], why-not top-k spatio-textual query [28], and time-aware spatio-textual query [27]. However, these works did not consider streaming objects.

Pub/Sub system. Recently, location-aware pub/sub systems have been studied [2], [4], [8], [9], [13], [14], [20], [21]. The works in [8], [13], [14], [21] studied the boolean matching problem. For example, in [21] and [14], AP-tree and FAST, which can find subscriptions such that the location of a newly generated object o is in their ranges and the keywords of o contain their keywords, were proposed. Moreover, literature [9] studied a similarity search problem in which each query has a pre-given threshold. These works differ from ours because they do not consider top-k monitoring. Literatures [2] and [20] addressed the problem of monitoring spatio-textual top-k objects for a large number of subscriptions. The methods proposed in these works maintain subscriptions with an index based on a Quad-tree and filter the subscriptions whose top-k results do not change when an object is generated. However, these works assume that the locations of users (subscriptions) are static. These methods cannot simply maintain moving subscriptions.

Moving query processing. Many studies have addressed the problem of monitoring spatio-textual objects for moving queries [8], [10], [12], [22], [23]. The works in [8], [10], [12], [23] proposed methods that compute the safe region of a query q and re-evaluate the result of q when the user who issued q exits his/her safe region. However, literatures [10], [12], [23] did not consider streaming objects. Because the safe region may change because of the generation of new objects, the methods proposed in these works need to re-compute the safe region whenever a new object is generated. This incurs a large evaluation cost. On the other hand, literatures [8], [22] considered streaming objects. For example, in [8], an impact region, which is a region where a newly generated object may become the result when a user is in his/her safe region, was proposed. However, these works differ from ours because they focus on range queries.

IX. Conclusion

Many applications based on pub/sub systems and moving query processing have been attracting a lot of attention. In this paper, we addressed the problem of monitoring top-k spatio-textual objects for a large number of moving top-k spatio-textual subscriptions. To tackle this problem, we proposed a novel system, called Lamps. Lamps employs a novel index, namely SQ-tree, which maintains subscriptions based on their safe regions and a Quad-tree, and it can filter subscriptions whose top-k results do not change when a new object is generated. Moreover, we proposed an efficient algorithm to retrieve the objects that become top-k objects when an object expires or when users exit their safe regions. We furthermore developed a novel approximation technique for safe region construction. We evaluated Lamps by experiments on real datasets, and the results show that Lamps achieves higher performance than baseline approaches.

Acknowledgment. This research was partially supported by the Grant-in-Aid for Scientific Research(A)(18H04095) and (B)(T17KT0082a) of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] L. Chen, G. Cong, and X. Cao. An efficient query indexing mechanism for filtering geo-textual data. In *SIGMOD*, pages 749–760, 2013.
[2] L. Chen, G. Cong, X. Cao, and K.-L. Tan. Temporal spatial-keyword top-k publish/subscribe. In *ICDE*, pages 255–266, 2015.
[3] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: an experimental evaluation. *PVLDB*, 6(3):217–228, 2013.
[4] L. Chen, S. Shang, K. Zheng, and P. Kalnis. Cluster-based subscription matching for geo-textual data streams. In *ICDE*, pages 890–901, 2019.
[5] R. Chuitian, L. Chunbin, N. S. Yasin, W. Jianguo, L. Wei, and D. Xiaoyong. Fast and scalable distributed set similarity joins for big data analytics. In *ICDE*, pages 1059–1070, 2017.
[6] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.
[7] L. Guo, J. Shao, H. H. Aung, and K.-L. Tan. Efficient continuous top-k spatial keyword queries on road networks. *GeoInformatica*, 19(1):29–60, 2015.
[8] 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.
[9] 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.
[10] 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.

[11] A. C. König, K. Church, and M. Markov. A data structure for sponsored search. In *ICDE*, pages 90–101, 2009.

[12] C. Li, Y. Gu, J. Qi, G. Yu, R. Zhang, and W. Yi. Processing moving knn queries using influential neighbor sets. *PVLDB*, 8(2):113–124, 2014.

[13] G. Li, Y. Wang, T. Wang, and J. Feng. Location-aware publish/subscribe. In *SIGKDD*, pages 802–810, 2013.

[14] A. R. Mahmood, A. M. Aly, and W. G. Aref. Fast: Frequency-aware indexing for spatio-textual data streams. In *ICDE*, pages 305–316, 2018.

[15] M.-H. Park, J.-H. Hong, and S.-B. Cho. Location-based recommendation system using bayesian user's preference model in mobile devices. In *UIC*, pages 1130–1139, 2007.

[16] J. Qi, R. Zhang, C. S. Jensen, K. Ramamohanarao, and J. HE. Continuous spatial query processing: A survey of safe region based techniques. *Computing Surveys*, 51(3):64, 2018.

[17] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *SSTD*, pages 205–222, 2011.

[18] K. Subbian, C. Aggarwal, and K. Hegde. Recommendations for streaming data. In *CIKM*, pages 2185–2190, 2016.

[19] X. Wang, W. Zhang, Y. Zhang, X. Lin, and Z. Huang. Top-k spatial-keyword publish/subscribe over sliding window. *The VLDB Journal*, 26(3):301–326, 2017.

[20] 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.

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

[22] 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.

[23] 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.

[24] B. Žalik. Two efficient algorithms for determining intersection points between simple polygons. *Computers & Geosciences*, 26(2):137–151, 2000.

[25] C. Zhang, Y. Zhang, W. Zhang, and X. Lin. Inverted linear quadtree: Efficient top k spatial keyword search. *TKDE*, 28(7):1706–1721, 2016.

[26] D. Zhang, C.-Y. Chan, and K.-L. Tan. Processing spatial keyword query as a top-k aggregation query. In *SIGIR*, pages 355–364, 2014.

[27] J. Zhao, Y. Gao, G. Chen, and R. Chen. Towards efficient framework for time-aware spatial keyword queries on road networks. *TOIS*, 36(3):24, 2018.

[28] J. Zhao, Y. Gao, G. Chen, and R. Chen. Why-not questions on top-k geo-social keyword queries in road networks. In *ICDE*, pages 965–976, 2018.

[29] 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*, pages 387–398, 2017.

[30] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, pages 155–162, 2005.