Noname manuscript No.
(will be inserted by the editor)

Top-k term publish/subscribe for geo-textual data streams

Lisi Chen · Shuo Shang · Christian S. Jensen · Jianliang Xu · Panos Kalnis · Bin Yao · Ling Shao

Received: date / Accepted: date

Abstract Massive amounts of data that contain spatial, textual, and temporal information are being generated at a rapid pace. With streams of such data, which includes check-ins and geo-tagged tweets, available, users may be interested in being kept up-to-date on which terms are popular in the streams in a particular region of space. To enable this functionality, we aim at efficiently processing two types of general top-k term subscriptions over streams of spatio-temporal documents: Region-based Top-k Spatial-Temporal Term (RST) subscriptions and Similarity-based Top-k Spatial-Temporal Term (SST) subscriptions. RST subscriptions continuously maintain the top-k most popular trending terms within a user-defined region. SST subscriptions free users from defining a region and maintain top-k locally popular terms based on a ranking function that combines term frequency, term recency, and term proximity. To solve the problem, we propose solutions that are capable of supporting real-life location-based publish/subscribe applications that process large numbers of SST and RST subscriptions over a realistic stream of spatio-temporal documents. The performance of our proposed solutions is studied in extensive experiments using two spatio-temporal datasets.

Keywords publish · subscribe · spatio-temporal · keyword · stream

Shuo Shang ✉
University of Electronic Science and Technology of China
E-mail: jedi.shang@gmail.com

Lisi Chen
University of Electronic Science and Technology of China
E-mail: lchen012@e.ntu.edu.sg

Christian S. Jensen
Aalborg University, Denmark
E-mail: csj@cs.aau.dk

Jianliang Xu
Hong Kong Baptist University
E-mail: xujl@comp.hkbu.edu.hk

Panos Kalnis
King Abdullah University of Science and Technology, Saudi Arabia
E-mail: panos.kalnis@kaust.edu.sa

Bin Yao
Shanghai Jiao Tong University, China
E-mail: yaobin@cs.sjtu.edu.cn

Ling Shao
Inception Institute of Artificial Intelligence, UAE
E-mail: ling.shao@ieee.org

1 Introduction

Very large volumes of spatio-temporal documents are being generated at a rapid pace by social media users. For example, Twitter has more than 300 million monthly active users who post 500 million tweets per day [69]. All tweets are associated with a timestamp that indicates their arrival time, and many tweets are associated with locations, which may be either coordinates (latitude and longitude) or semantic locations (e.g., "Chicago, IL, U.S.A."). Beyond Twitter, location-based social networking services (e.g., Foursquare, Yelp, Booking.com) allow users to publish check-ins or reviews that contain text descriptions, geographical information, and timestamps.

Such spatio-temporal documents that arrive continuously in data streams often offer first-hand information about local events of different types and scales [27]. In particular, many local breaking news stories and other matters of public interest were first reported through tweets [23, 27, 30, 32]. Additionally, we can get the gist of many other timely and informative news items and events, such as business promotions, comments, and reviews, through tweets [55].

Due to the increasing proliferation of geo-textual streams, the problem of developing location-based publish/subscribe systems that can support large numbers of subscribers, allowing them to continuously receive spatio-temporal documents relevant to their subscriptions, has received substantial attention (e.g., [10, 11, 25, 28, 33, 60, 68, 70]). The feeding pattern of these location-based publish/subscribe systems for spatio-temporal document streams is *keyword and location driven* and *item-based*, meaning that (1) subscribers need to define both subscription keywords and subscription locations, and that (2) subscribers continuously receive documents that satisfy their keyword and location subscriptions. This type of publish/subscribe system is useful and effective when subscribers know exactly what they are looking for and are able to define their intentions in the form of precise keywords to obtain targeted results. However, in many situations, subscribers may not have, or may not provide, clear intentions. As pointed out in the literature [6, 65], web users are often unwilling to spend extra effort on specifying their intentions. Further, even if they are motivated to doing so, they may often be unsuccessful at doing so [65]. As a result, going beyond location and keyword search, many users wish to know what is happening around them by receiving an up-to-date timely overview that captures local bursty events, trending topics, public concerns, and what occupies the minds of local users. This can be achieved by finding frequent or bursty terms in nearby documents (e.g., [3, 40, 41, 56]).

Inspired by this, we take the first step towards developing a location-aware *term* publish/subscribe system for geo-textual data streams. Specifically, the system supports location-based term subscriptions and continuously maintains top-k locally popular terms that occur in a stream of spatio-temporal documents for each subscription. We take term frequency, term freshness, and the location proximity between term and subscription into consideration when quantifying the popularity of a term.

Delivering terms to subscribers has the following benefits: First, since the top-k most locally popular terms are inclined to cover the most significant topics that occupy the minds of local users, the content of a spatio-temporal documents published in the region can be expressed in an informative and concise way by applying visualization techniques (e.g., “Word Clouds”). Second, with top-k term search, terms from near-duplicate documents are likely to be merged. Thus, a subscriber will not suffer from receiving many near-duplicate messages. Third, top-k term subscriptions free subscribers from specifying keywords and other difficult-to-set parameters.

In this paper, we develop the location-based term publish/subscribe system that is able to processes a large number of location-based term subscriptions over a stream of spatio-temporal documents. We define and study two types of term subscriptions: R̲egion-based Top-k S̲patial-Temporal T̲erm (RST) subscriptions and S̲imilarity-based Top-k S̲patial-Temporal T̲erm (SST) subscriptions. Specifically, RST subscriptions continuously maintain the top-k most popular trending terms within a user-defined region. This kind of subscription relies on a *temporal popularity score* that quantifies the popularity of a term by taking the following two aspects into account: (1) The frequency of the term in documents published in the subscription region; (2) The recency of documents that contain the term and were published within the subscription region. The RST subscriptions are useful when subscribers want to apply a hard spatial constraint on the input data streams [10, 19, 33, 61, 67], For example, a subscriber living in Chicago may only be interested in exploring spatio-temporal documents published in the urban region of Chicago. Unlike RST subscriptions, SST subscriptions free subscribers from specifying a region. In particular, SST subscriptions maintain top-k locally popular terms based on a *spatio-temporal popularity score* that combines term frequency, term recency, and term spatial proximity (i.e., the aggregated spatial proximities between the subscription location and the spatio-temporal documents that contain the term). SST subscriptions are useful when subscribers have local intent but do not have a hard spatial constraint (see Section 5.4 for use cases of RST and SST subscriptions). We aim at maintaining the up-to-date top-k terms for a large number of SST and RST subscriptions over a stream of spatio-temporal documents with real-life arrival rate.

Framework overview: Figure 1 presents the framework for processing location-based term subscriptions over a stream of spatio-temporal documents. The input consists of two parts: (1) spatio-temporal documents published by location-based social media; (2) Term subscriptions (i.e., SST and RST subscriptions) registered by users. When a new spatio-temporal document is published, we first decompose the document into three components: a bag of terms, a location, and a timestamp. Next, we retrieve the subscriptions whose top-k term lists must be updated given the new document. This step is called *subscription matching*. Finally, we update their top-k term lists. During the subscription matching, we need to compute the updated popularity score between each subscription and each term in the new document, which is very time consuming (see Section 3.2 for time complexity analysis). Hence, we address the following three challenges in the subscription matching and result update processes:

– *Real-time maintenance of top-k result:* Unlike the term frequency, the term popularity score (including both temporal term popularity and spatio-temporal term popularity) takes time into consideration, which changes continuously. Consequently, existing solutions for the heavy-hitter problem (e.g., SpaceSaving [42] and Lossy-Counting [39]) cannot be applied to our problem directly. We need to develop an efficient approach to maintaining the top-k terms of each subscription given the time-dependent, continuously changing popularity score in terms of each subscription and each term.

- *Efficient computation of spatio-temporal term popularity:* When processing SST subscriptions, as a new document arrives, we need to update the spatio-temporal term popularity score of each matched subscription, which involves frequent computations of the popularity score of a term w.r.t. a subscription. This popularity score computation is expensive because we must take into account the current time and all documents containing the term (see Section 3.3 for time complexity analysis). Therefore, we provide means of lowering the complexity and time cost of term popularity computations.
- *Group filtering of subscription matching:* Publish/subscribe settings are characterized by large numbers of subscriptions. Therefore, when finding the subset of subscriptions that "match" [1] a term from a new document, we would like to consider each subscription individually. Specifically, we propose to group similar subscriptions and develop two effective subscription group filtering mechanisms for processing SST and RST subscriptions so that unqualified subscriptions can be eliminated at low cost.

The present paper expands on a previous study [17]. Specifically, we define and study a new category of location-based term subscription, Similarity-based Top-k Spatial-Temporal Term (SST) subscription. Compared with the RST subscriptions covered in the previous study [17], SST subscriptions free users from specifying query regions by adopting a general ranking metric, spatio-temporal popularity, that combines term frequency, term recency, and term proximity. Efficient processing of SST subscriptions incurs additional challenges: Unlike RST subscriptions that directly filter spatio-temporal documents located outside their query regions, SST subscriptions have to regard all spatio-temporal documents as matching candidates. To address this challenge, we develop a new subscription matching algorithm (Section 3.2) and a new group filtering mechanism (Section 3.4). Next, it is expensive to compute spatio-temporal popularity between each pair of a subscription and a term. We propose an efficient method to compute tight bounds for spatio-temporal popularity (Section 3.3). Additionally, we conduct extensive experiments to evaluate the performance of baselines and our proposal for processing large numbers of SST subscriptions over streams of spatio-temporal documents (Section 5.2).

To address these challenges, we propose an approach that exploits the following techniques to process RST and SST subscriptions.

[1] We say a subscription matches a term if the term is a top-k result of the subscription

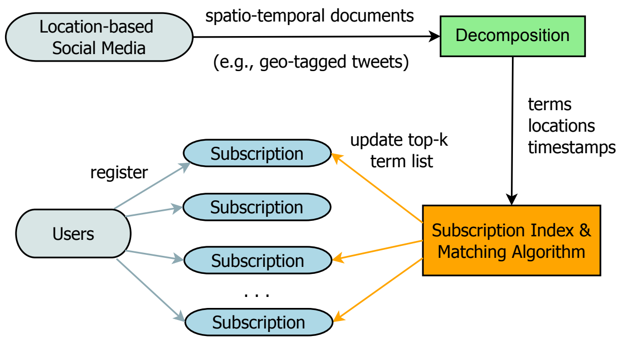


Fig. 1 Framework for processing SST and RST subscriptions

(1) We propose approaches that enable efficient computation of real-time top-k result updates of SST and RST subscriptions by avoiding the popularity score re-computation of each term in a subscription, and thus reduce the time complexity (Sections 3.2 and 4.3).

(2) We propose a novel algorithm to efficiently compute spatio-temporal term popularity when processing SST subscriptions (Section 3.3).

(3) We develop subscription grouping and filtering techniques that enable effective group pruning of unqualified subscriptions during subscription matching (Sections 3.3.2 and 4.4).

To sum up, the paper's contributions are twofold. First, we define the SST and RST subscriptions and present the first study on the problem of maintaining the up-to-date *terms* for a large number of term subscriptions over a stream of spatio-temporal documents. Second, we develop efficient solutions comprising the key techniques mentioned above to process large numbers of SST and RST subscriptions. Our experimental results suggest that our proposal is able to achieve reductions of the processing time by 70%–95% and 60%– 90% for processing SST subscriptions and RST subscriptions, respectively, when compared with baselines developed based on existing techniques. Further, the results suggest that our proposal is capable of supporting 2.1 million SST subscriptions or 1.8 million RST subscriptions over a geo-textual data stream with a real-life arrival rate [57].

2 Problem Statement

We define the notion of a spatio-temporal document and then define the Similarity-based Top-k Spatial-Temporal Term (SST) subscription and the Region-based Top-k Spatial-Temporal Term (RST) subscription.

Definition 1: *Spatio-temporal document.* A spatio-temporal document is defined as a triple $d = \langle \psi, \rho, t_c \rangle$, where ψ is a set of keywords, ρ is a point location with latitude and longitude, and t_c is the document's creation time. □

We consider a stream of spatio-temporal documents. For example, a stream can be tweets with geographical information (i.e., geo-tagged tweets), check-ins with text descriptions in Foursquare, and web pages with locations.

Note that freshness is important in spatio-temporal streams. For example, tweets often capture events, the significance of which decline as time elapses [55]. Next, we define SST and RST subscriptions.

2.1 SST Subscription

Definition 2: *Similarity-based Top-k Spatial-Temporal Term* (SST) *Subscription.* An SST subscription is denoted by $s = \langle \rho, k\rangle$, where ρ is a subscription location (i.e., a geographical point location with latitude and longitude) and k defines the number of popular terms to be maintained. An SST subscription continuously feeds users with new terms whose *spatio-temporal popularity score* are ranked in the top-k. □

Specifically, the spatio-temporal popularity of a term w with regards to an SST subscription s at time t is computed as follows:

$$\mathsf{LTP}(s,w,t) = \sum_{d \in U} \mathsf{SF}(s,d,w) \cdot D^{-(t-d.t_c)}, \tag{1}$$

where U denotes the document collection (existing spatio-temporal documents) and $\mathsf{SF}(s,d,w)$ can be any function that satisfies the following requirements: (1) $\mathsf{SF}(s,d,w)$ is monotonically increasing with regards to the frequency of w and (2) $\mathsf{SF}(s,d,w)$ is monotonically decreasing with regards to the spatial proximity between s and d. Expression $D^{-(t-d.t_c)}$ is an exponential decaying function [34] that favors the terms in more recent documents. Specifically, D is the base that indicates the decaying rate, and the function is monotonically decreasing with $t-d.t_c$. This function is used widely (e.g., [5,9,35,55]) to measure the recency of items in stream data. Experimental studies suggest that the exponential decaying function is effective in blending information freshness and text relevancy [22]. Without loss of generality, we use the following ranking function to compute the LTP score:

$$\mathsf{LTP}(s,w,t) = \sum_{d \in U} \frac{F(d,w) \cdot D^{-(t-d.t_c)}}{\alpha + dist(s,d)} \tag{2}$$

where $F(d,w)$ is the frequency of term w in document d, t is the current time, $dist(\cdot)$ denotes Euclidean distance, and α is a parameter that enables control of the weight of the spatial proximity. In particular, a larger value of α indicates lower emphasis on spatial proximity.

2.2 RST Subscription

We proceed to define the Region-based Top-k Spatial-Temporal Term (RST) subscription. Unlike SST, RST requires a subscriber to define a spatial region rather than a location.

Definition 3: *Region-based Top-k Spatial-Temporal Term* (RST) *Subscription.* An RST subscription is denoted by $s = \langle r, k\rangle$, where r is a subscription region (i.e., a circle or a rectangle), and k is the number of frequent terms to be maintained. A subscription aims to continuously return new terms whose temporal popularities are ranked within the top-k based on the spatio-temporal documents published within a subscription region. □

The temporal popularity of a term w with regard to an RST subscription s is defined as follows:

$$\mathsf{TP}(s,w,t) = \sum_{d \in s.r} F(d,w) \cdot D^{-(t-d.t_c)} \tag{3}$$

Based on our publish/subscribe scenario, the arrival rate of spatio-temporal documents (e.g., tweets) is at the scale of millions a day [57], while new subscriptions are arriving at the rate of tens of thousands a day, and we may support and serve millions of subscriptions. Hence, we aim to develop a scalable solution to maintain up-to-date results for a large number of SST and RST subscriptions over a data stream of spatio-temporal documents.

3 SST Subscription Processing

This section presents a baseline and our solution for processing a large number of SST subscriptions over a stream of spatio-temporal documents.

3.1 Baseline Solution

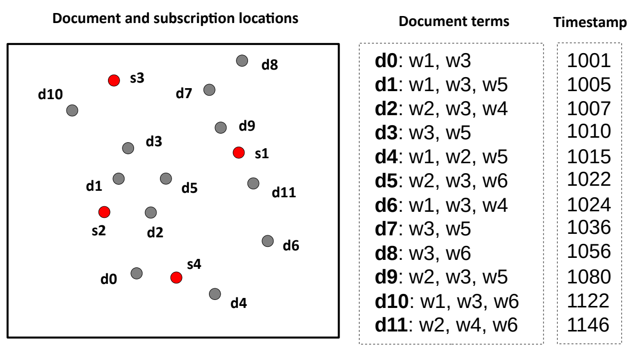
3.1.1 Subscription matching and result update

We maintain an exact top-k term list for each SST subscription s in real-time fashion according to the LTP score. When a new document d arrives, we re-compute $\mathsf{LTP}(s,w_i,t)$ for each term $w_i \in d.\psi$ and each subscription s in the subscription pool. Then we update the top-k term list maintained by each subscription. According to Equation 2, if an existing document d_e does not contain w_i (i.e., $F(d_e,w_i) = 0$) we may simply skip d_e because d_e does not contribute to $\mathsf{LTP}(s,w_i,t)$. Hence, to compute $\mathsf{LTP}(s,w_i,t)$, we just need to retrieve and evaluate documents that contain w_i. To facilitate the retrieval of documents containing a particular term, we maintain an inverted file. To maintain the top-k term list for an SST subscription s, we adopt a min-heap for s in

Document	Distance to s_1	Ordered result
d_0	0.50	w_1, w_3
d_1	0.35	w_1, w_3, w_5
d_2	0.30	w_3, w_1, w_5
d_3	0.30	w_3, w_5, w_1
d_4	0.45	w_5, w_3, w_1
d_5	0.20	w_3, w_2, w_5
d_6	0.25	w_3, w_2, w_1
d_7	0.20	w_3, w_5, w_2
d_8	0.30	w_3, w_5, w_2
d_9	0.10	w_3, w_2, w_5
d_{10}	0.60	w_3, w_2, w_5
d_{11}	0.15	w_3, w_2, w_5

Table 1 Result updates of s_1

which term w is represented by $w = \langle id, p, t\rangle$, where id denotes the entry (ID) of term w, p indicates the LTP score at the time of the last update, and t denotes the timestamp of the last update. The elements in a min-heap are sorted by $w.p \cdot D^{-(t_{cur}-w.t)}$, where t_{cur} denotes the current time. To ensure that elements in a min-heap are sorted correctly, we need to re-sort the elements because $w.p \cdot D^{-(t_{cur}-w.t)}$ changes as time passes.



Document terms	Timestamp
d0: w1, w3	1001
d1: w1, w3, w5	1005
d2: w2, w3, w4	1007
d3: w3, w5	1010
d4: w1, w2, w5	1015
d5: w2, w3, w6	1022
d6: w1, w3, w4	1024
d7: w3, w5	1036
d8: w3, w6	1056
d9: w2, w3, w5	1080
d10: w1, w3, w6	1122
d11: w2, w4, w6	1146

Fig. 2 Subscriptions and a stream of documents

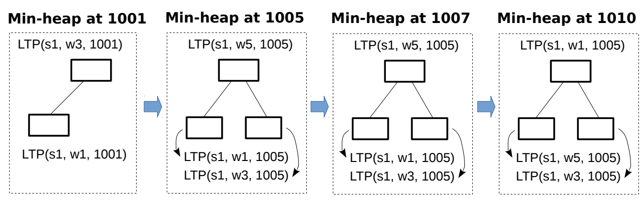


Fig. 3 Evolving min-heap for s_1

Example 1 Figure 2 presents an example, where red circles (s_1–s_3) denote the SST locations of subscriptions and gray circles are the locations of the documents from the stream (d_0–d_{11}). The document terms and timestamps are listed to the right. Table 1 presents the updates to the ordered result list for s_1 that occur as documents arrive. The result list is a min-heap sorted by the LTP score (cf. Figure 3).

3.1.2 Document and subscription updates

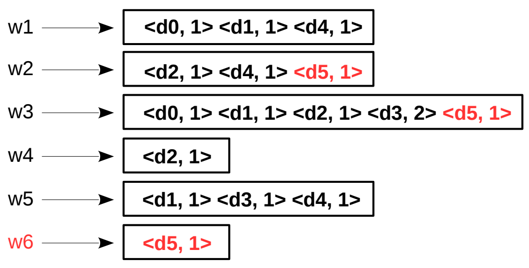

w1 <d0, 1> <d1, 1> <d4, 1>
w2 <d2, 1> <d4, 1> <d5, 1>
w3 <d0, 1> <d1, 1> <d2, 1> <d3, 2> <d5, 1>
w4 <d2, 1>
w5 <d1, 1> <d3, 1> <d4, 1>
w6 <d5, 1>

Fig. 4 Document inverted file between timestamps 1015 and 1022

Document insertion: When a new document d arrives, we index d by an inverted file. Figure 4 shows the inverted file between timestamps 1015 and 1022 based on the document stream from Figure 2. Each posting in a posting list (e.g., $\langle d_3, 2\rangle$ in the posting list for term w_3) consists of two elements: the document id (i.e., d_3) and the frequency of the term of the posting list in the document (i.e., 2 occurrences of w_3 in d_2). The black components in Figure 4 were constructed before timestamp 1015, and the red components are inserted at timestamp 1022 when d_5 arrives.

Document deletion: We utilize a *lazy removal* strategy to delete outdated documents from the inverted file. Specifically, we remove as much outdated documents as possible while guaranteeing that the accuracy of the LTP score is bounded by an error threshold Θ_{err}. Before presenting our document deletion scheme, we introduce two concepts: *maximum error* (Lemma 1) and *minimum LTP* (Lemma 2).

Lemma 1 *Given a term w, a timestamp t, and a document d, for any* SST *subscription s, the maximum error of* LTP(s, w, t) *incurred by removing the posting of d from the posting list for w is computed as follows:*

$$Err_{max}(w, d) = \frac{F(d, w) \cdot D^{-(t-d.t_c)}}{\alpha} \tag{4}$$

Proof: Based on Equation 2, the actual error of LTP(s, w, t) *incurred by removing the posting of d from the posting list for w is $\frac{F(d,w)\cdot D^{-(t-d.t_c)}}{\alpha+dist(s,d)}$. Because $\alpha \leq \alpha + dist(s, d_i)$, it follows that $Err_{max}(w, d)$ never underestimates the actual error. This completes the proof.* □

Lemma 2 *Let d_{max} be the maximum possible distance in the underlying space. Given a term w, a timestamp t, and a document d, for any* SST *subscription s, the minimum possible LTP score* LTP$_{min}(w, t)$ *is computed as follows:*

$$\mathsf{LTP}_{min}(w, t) = \sum_{d_i \in IL_w} \frac{F(d_i, w) \cdot D^{-(t-d_i.t_c)}}{\alpha + d_{max}} \tag{5}$$

Here, IL_w denotes the posting list for term w.

Proof: *Given a subscription s, the exact LTP score $\mathsf{LTP}(s,w,t)$ is computed by Equation 2. Because $\alpha + d_{max} \geq \alpha + dist(s,d_i)$, we can deduce that $\mathsf{LTP}_{min}(w,t)$ never exceeds $\mathsf{LTP}(s,w,t)$. This completes the proof.* □

After traversing the posting list IL_w during subscription matching, we remove the top-n oldest document postings that occur in IL_w. In particular, n is the maximum value satisfying the following condition:

$$\frac{Err_{acc}(w)\cdot D^{-(t-t_w)} + \sum_{i=1}^{n} Err_{max}(w,d_i)}{\mathsf{LTP}_{min}(w,t)} \leq \Theta_{err} \quad (6)$$

Here, Θ_{err} is an error-rate threshold that guarantees that the error rate of the LTP score incurred by deleting the n oldest postings does not exceed Θ_{err}, and $Err_{acc}(w)$ denotes the accumulated error at the timestamp t_w of last deletion operation on IL_w. Each time we perform document deletion, we update $Err_{acc}(w)$ and t_w.

Subscription insertion and deletion The baseline solution does not have an index structure for SST subscriptions. Each time a new subscription is registered, we add the subscription to the subscription list. Similarly, if a user requests to de-register a subscription, we simply remove it from the subscription list.

Time complexity: Recall that processing of a new document d involves two steps:

(1) *Subscription matching:* We compute the updated LTP score of each term w.r.t. each subscription and find a subset of subscriptions that match d. The complexity of subscription matching is $O(|d.\psi|\cdot|IL_{d.\psi}|\cdot|S|\cdot C_{SF})$, where $|S|$ denotes the number of existing SST subscriptions, $|IL_{d.\psi}|$ is the number of documents (postings) in the posting lists associated with terms in d (cf. Equation 7), and C_{SF} denotes the complexity of computing the SF score (cf. Equation 1), which is generally $O(1)$ but depends on the distance metric we use[2].

$$|IL_{d.\psi}| = \sum_{w_i \in d.\psi} |IL_{w_i}|, \quad (7)$$

where IL_{w_i} denotes the posting list for w_i.

(2) *Result update:* After subscription matching, we reorder the min-heap maintained by each matched subscription, which has time complexity $O(|S|\cdot k \cdot \log k)$, where k denotes the number of result terms maintained by each subscription.

(3) *Document update:* Each new document d is indexed by an inverted file. Hence, the time complexity of indexing d is $O(|d.\psi|)$.

[2] the complexity is $O(1)$ when the SF score is computed based on Euclidean distance or network distance with pre-computation of pair distances

(4) *Subscription update:* The baseline solution does not index SST subscriptions. Thus, the time complexities of subscription insertion and deletion are $O(1)$.

3.2 Subscription Processing with Tailored Result Update

3.2.1 Subscription matching

We proceed to introduce a tailored result update method and corresponding subscription matching algorithm, which are able to reduce the time complexity of result update to $O(|S|\cdot \log k)$ while the time complexity of subscription matching is unchanged. Recall that it is computationally expensive to re-order the min-heap maintained by each subscription. To address that problem, we introduce Lemma 3, which lays foundation for preventing such re-computations.

Lemma 3 *Let s be an SST subscription, w and w' be two different terms. If $\mathsf{LTP}(s,w,t) > \mathsf{LTP}(s,w',t)$ and no document contains w or w' arrives during the time period $[t, t+\Delta t]$ ($\Delta t > 0$), we have:*

$$\mathsf{LTP}(s,w,t+\Delta t) > \mathsf{LTP}(s,w',t+\Delta t).$$

Proof: *According to Equation 2 we have:*

$$\mathsf{LTP}(s,w,t) > \mathsf{LTP}(s,w',t) \iff$$
$$\sum_{d\in U}\frac{F(d,w)\cdot D^{-(t-d.t_c)}}{\alpha + dist(s,d)} > \sum_{d\in U}\frac{F(d,w')\cdot D^{-(t-d.t_c)}}{\alpha + dist(s,d)}.$$

Consequently, we have:

$$D^{\Delta t}\cdot\sum_{d\in U}\frac{F(d,w)\cdot D^{-(t+\Delta t-d.t_c)}}{\alpha + dist(s,d)} >$$
$$D^{\Delta t}\cdot\sum_{d\in U}\frac{F(d,w')\cdot D^{-(t+\Delta t-d.t_c)}}{\alpha + dist(s,d)}$$

This completes the proof. □

Lemma 3 indicates that the LTP score of a term in an SST subscription has the following property: the relative ranking of two different terms w.r.t. a subscription is consistent over time. Hence, the min-heap maintained by each subscription will remain correct and we do not need to re-evaluate the k terms in the min-heap for each subscription when each new document arrives.

When a new document d_n arrives, for each term w_i in $d_n.\psi$ we traverse its posting list in the inverted file and retrieve the documents containing w_i to compute $\mathsf{LTP}(s,w_i,t_{cur})$. At the same time, we compare $\mathsf{LTP}(s,w_i,t_{cur})$ with the current LTP score of the top document in the min-heap, $\mathsf{LTP}(s,w_{min},t_{cur})$. The LTP score of the top document stored in the min-heap was calculated

Algorithm 1: TSubMatching

```
Input: Inverted lists $IL$, subscription set $S$, min-heaps $H[s]$ of each subscription $s$, new document $d$
Output: Updated inverted lists $IL$, updated min-heaps $H[s]$ of each subscription $s$
1  for each unique term $w_i$ in $d$ do
2      $IL[w_i]$.add($d$);
3      for each $s$ in $S$ do
4          $P_{new} \leftarrow 0$;
5          for each posting $d_j$ in $IL[w_i]$ do
6              $P_{new} \leftarrow P_{new} + F(d, w_i) \cdot D^{-(t-d_j.t_c)}/(\alpha + dist(s, d_j))$;
7          $w_t \leftarrow H[s].top()$;
8          $P_e \leftarrow w_t.p$;
9          $t_e \leftarrow w_t.t$;
10         $P_{cur} \leftarrow P_e \cdot D^{-(t_{cur}-t_e)}$;
11         if $H[s]$ contains $w_i$ then
12             $H[s]$.delete($w_i$);
13             $H[s]$.push($\langle w_i, P_{new}, t_{cur}\rangle$);
14         else if $P_{new} > P_{cur}$ then
15             $H[s]$.pop();
16             $H[s]$.push($\langle w_i, P_{new}, t_{cur}\rangle$);
17 return;
```

at a previous timestamp t_e (i.e., as $\mathsf{LTP}(s, w_{min}, t_e)$). So we need to compute the LTP score as follows:

$$\mathsf{LTP}(s, w_{min}, t_{cur}) = \mathsf{LTP}(s, w_{min}, t_e) \cdot D^{-(t_{cur}-t_e)} \quad (8)$$

If $\mathsf{LTP}(s, w_i, t_{cur}) > \mathsf{LTP}(s, w_{min}, t_{cur})$, we update the min-heap maintained by s.

Before presenting our algorithm of subscription matching, we define the concept of Partial LTP score, which computes the portion of an LTP score contributed by a given document.

Definition 4: *Partial LTP score.* Given a document d, a term w, a timestamp t, and an SST subscription s, the partial LTP score of d with respect to s, denoted by $\mathsf{P}(s, w, t, d)$, is defined as follows:

$$\mathsf{P}(s, w, t, d) = \frac{F(d, w) \cdot D^{-(t-d.t_c)}}{\alpha + dist(s, d)}$$

□

Algorithm 1 presents the pseudo code of our document processing algorithm, which includes matching and result update. The inputs are a new document d, a subscription set S, a min-heap $H[s]$ maintained for each subscription s in S, and the document inverted lists IL. The outputs are updated inverted lists IL and an updated min-heap $H[s]$ of each subscription s. When a new spatio-temporal document d arrives, we process each unique term w_i in d (Line 1). In particular, we first add d (i.e., the posting of d) to the inverted list of term w_i (Line 2). Next, we evaluate each SST subscription s (Lines 3–16). Here, we compute the LTP score of w_i in terms of s (Lines 5–9) and check whether w_i is a top-k result of s (Lines 10–16). Specifically, we first initialize the LTP score of w_i in terms of s (i.e., P_{new}) to 0 (Line 4). Then we traverse the postings list of term w_i (i.e., $IL[w_i]$) (Lines 5–6). For each posting d_j in $IL[w_i]$ we calculate its partial LTP score contributed by P_{new} (i.e., $\frac{F(d,w_i)\cdot D^{-(t-d_j.t_c)}}{(\alpha+dist(s,d_j))}$) and sum them up based on Equation 2 (Line 6). Having computed P_{new}, we visit the min-heap for s and retrieve its top element w_t (Line 7). Here, P_e denotes the LTP score of w_t at the time of the last update, and t_e denotes the timestamp of last update (Lines 8–9). Next, we compute the LTP score of w_t at the current timestamp (P_{cur}) using Equation 8 (Line 10). Note that if w_i exists in min-heap $H[s]$, we need to update w_i to reflect the new LTP score (Lines 11–13). If the current LTP score of w_i exceeds that of w_t, we pop w_t and push w_i onto the min-heap (Lines 14–16). This completes the update of the top-k result of s, which is maintained by the min-heap. Theorem 1 characterizes the time complexity of TSubMatching.

Theorem 1 *Given a new spatio-temporal document d, a set of existing documents organized by inverted file IL, a set of existing* SST *subscriptions S, and a result cardinality k, the time complexity of* TSubMatching *is $O(|S| \cdot |d.\psi| \cdot |IL_{d.\psi}| \cdot C_{SF} + |S| \cdot \log k)$, where $|IL_{d.\psi}|$ is defined by Equation 7.*

Proof: TSubMatching *evaluates each unique term in $d.\psi$. For each such term, its inverted list is traversed to calculate the LTP score for the term and every* SST *subscription in S. If the term is a top-k result of a subscription, we need to update its min-heap, which requires $O(\log k)$.* □

3.2.2 Document and subscription updates

Like the baseline solution, subscription processing with tailored result update does not require a dedicated indexing structure for SST subscriptions. So the time complexity for subscription insertion and deletion is $O(1)$. The document insertion and deletion operations are the same as baseline. So the complexity of document update is $O(|d.\psi|)$.

3.3 Fast LTP Score Computation

According to Theorem 1, the complexity of Algorithm 1 is $O(|d.\psi| \cdot |IL_{d.\psi}| \cdot |S| + |S| \cdot \log k)$. Here, $|IL_{d.\psi}|$ can be very large, and it is time consuming to compute the LTP score for each posting (document) in the inverted file. To address this problem, we develop an online partitioning method that groups documents in posting lists. Based on the partitioning result, we compute the *aggregate partial LTP score* of a term w in an SST subscription s contributed by a document set (group) in the posting list.

Definition 5: *Aggregated partial LTP score.* Given a group of documents G associated with a postings list of term w

and an SST subscription s, the aggregate partial LTP score, denoted by $\mathsf{AP}_G(s, w, t)$, is defined as follows:

$$\mathsf{AP}_G(s, w, t) = \sum_{d \in G} \frac{F(d, w) \cdot D^{-(t - d.t_c)}}{\alpha + dist(s, d)} \tag{9}$$

□

The LTP score can be computed by summing the aggregate partial LTP score of each group:

$$\mathsf{LTP}(s, w, t) = \sum_{G \in \mathcal{G}} \mathsf{AP}_G(s, w, t), \tag{10}$$

where $\mathcal{G}$ denotes the group set associated with term w. Here, the challenge is that without evaluating each document in G, we cannot compute an exact value of $\mathsf{AP}_G(s, w, t)$. Instead, it is possible to derive an upper bound on $\mathsf{AP}_G(s, w, t)$, which are presented in Equation 11.

$$\mathsf{AP}_G(s, w, t).ub = \frac{\sum_{d \in G} F(d, w) \cdot D^{-(t - d.t_c)}}{\alpha + minDist(s, G)} \tag{11}$$

Above, $minDist(s, G)$ denotes the minimum Euclidean distance between s and spatio-temporal documents in G. The following two challenges exist when computing $\mathsf{AP}_G(s, w, t).ub$:

(1) *Efficiency:* Based on Equation 11, we still need to calculate $F(d, w) \cdot D^{-(t - d.t_c)}$ for each document in G, which has complexity $O(|G|)$. To avoid such one-by-one calculation, we aim to compute $\sum_{d \in G} F(d, w) \cdot D^{-(t - d.t_c)}$ with complexity $O(1)$ (cf. Section 3.3.1).

(2) *Effectiveness:* To generate a tighter bound, we need to compute a more accurate value of $minDist(s, G)$. We propose an approach to grouping documents in an inverted list based on online clustering. With this grouping approach, the value of $minDist(s, G)$ can be computed effectively (cf. Section 3.3.2).

3.3.1 Aggregate partial LTP score computation

According to Equations 11, $D^{-(t - d.t_c)}$ is different for each document. To enable group computation, we unify the value of $D^{-(t - d.t_c)}$ in each group. Specifically, for each group G we store the *aggregate popularity score* (cf. Equation 12) of the group at timestamp $G.t_{min}$, which denotes the creation time of the first document in G.

$$\mathsf{A}_{pop}(G) = \sum_{d \in G} F(d, w) \cdot D^{-(G.t_{min} - d.t_c)} \tag{12}$$

Note that the value of $\mathsf{A}_{pop}(G)$ stays constant as time elapses. We can easily calculate $\mathsf{AP}_G(s, w, t).ub$ based on $\mathsf{A}_{pop}(G)$ by Equation 13.

$$\mathsf{AP}_G(s, w, t).ub = \frac{\mathsf{A}_{pop}(G) \cdot D^{-(t - G.t_{min})}}{\alpha + minDist(s, G)} \tag{13}$$

Thus, the time complexity of computing $\mathsf{AP}_G(s, w, t).ub$ is reduced from $O(|G|)$ to $O(1)$.

3.3.2 Cluster-based document grouping

To improve the pruning power, we derive tighter lower and upper bounds on $\mathsf{AP}_G(s, w, t)$. Recall that $\mathsf{A}_{pop}(G)$ is an exact result. Based on Equation 13, $minDist(s, G)$ is the only variable that influences the accuracy of $\mathsf{AP}_G(s, w, t).lb$ (resp. $\mathsf{AP}_G(s, w, t).ub$). Thus, we propose to increase the accuracy of $minDist(s, G)$.

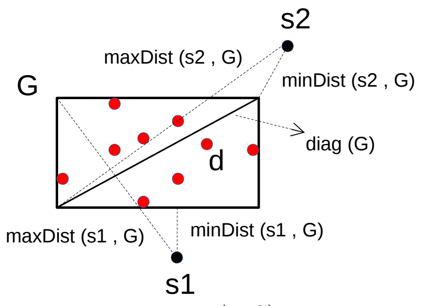


Fig. 5 Demonstration of $minDist(s, G)$

Figure 5 illustrates the values of $maxDist(s, G)$ and $minDist(s, G)$. Here, we adopt a minimum bounding rectangle (MBR) to represent the locations of spatio-temporal documents (red nodes) in group G. Let $diag(G)$ be the diagonal length of G's MBR. According to the triangle inequity, for any document d in G we have: $dist(s, d) - minDist(s, G) \leq diag(G)$. As a result, $diag(G)$ can be regarded as the maximum possible discrepancy between $minDist(s, G)$ and $dist(s, d)$. To increase the accuracy of $minDist(s, G)$, we constrain the value of $diag(G)$ for each group. For this purpose, we develop a Group-Constrained Online Clustering (GCOC) algorithm for clustering (grouping) documents in an inverted list such that the MBR of documents in each cluster (group) G satisfies $diag(G) \leq T_{dist}$, where T_{dist} is a pre-defined distance threshold. The high-level idea of GCOC algorithm is explained by Example 2.

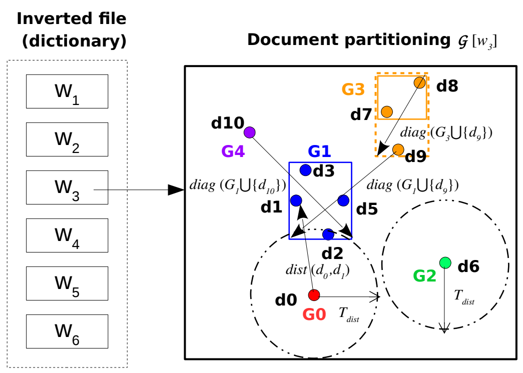


Fig. 6 Example of the GCOC algorithm

Example 2 Figure 6 depicts the spatio-temporal documents in the inverted list for w_3, and Table 2 presents the group-

	G_0	G_1	G_2	G_3	G_4
d_0	d_0				
d_1	d_0	d_1			
d_2	d_0	d_1, d_2			
d_3	d_0	d_1, d_2, d_3			
d_4	d_0	d_1, d_2, d_3			
d_5	d_0	d_1, d_2, d_3, d_5			
d_6	d_0	d_1, d_2, d_3, d_5	d_6		
d_7	d_0	d_1, d_2, d_3, d_5	d_6	d_7	
d_8	d_0	d_1, d_2, d_3, d_5	d_6	d_7, d_8	
d_9	d_0	d_1, d_2, d_3, d_5	d_6	d_7, d_8, d_9	
d_{10}	d_0	d_1, d_2, d_3, d_5	d_6	d_7, d_8, d_9	d_{10}

Table 2 Group updates

ing result when each new document (Column 1) arrives. Assume that document ID are assigned in chronological order. Each color of a document denotes a group. At first, d_0 arrives and forms a group (i.e., G_0). Next, d_1 arrives, and we compute the diagonal length of the MBR that contains d_0 and d_1, which is equivalent to the distance between d_0 and d_1. Because $dist(d_0, d_1) > T_{dist}$, we assign d_1 to a new group G_1. Note that if we assign d_0 and d_1 to the same group, the diagonal length of the MBR containing d_0 and d_1 will exceed T_{dist}. When d_2 arrives, we compute $dist(d_2, d_0)$ and $dist(d_2, d_1)$. Since $dist(d_2, d_1) < dist(d_2, d_0)$ and $dist(d_2, d_1) \le T_{dist}$, we assign d_2 to the group of d_1. We skip the processing of d_3–d_8[3]. When d_9 arrives, we compute the diagonal lengths of the MBRs for existing group (i.e., G_0–G_4) with d_9 included. We find that G_3 has the shortest diagonal length. So we further check if $G_3 \cup \{d_9\}$'s MBR satisfies the T_{dist} constraint. Because the constraint holds, we assign d_9 to G_3. We proceed to consider d_{10}. Group G_1 has the shortest diagonal length when including d_9. However, we find that $Diag(G_1 \cup \{d_{10}\}) > T_{dist}$, so we assign d_{10} to a new group (i.e., G_4).

Algorithm 2 presents the pseudo code of the GCOC algorithm. It takes the document partitioning list $\mathcal{G}$ as input. When a new document d arrives, we perform online grouping on the partitioning associated with each term w_i (i.e., $\mathcal{G}[w_i]$) (cf. Figure 6). In particular, we assign d to the group with MBR that has the shortest diagonal length. We first initialize G_{min} and $diag_{min}$, which denote the current group with the shortest diagonal length and the corresponding diagonal length, respectively (Lines 2–3). Next, for each group G in partitioning $\mathcal{G}[w_i]$ we add d to G, which is denoted by G' (Line 5). We find the group G_{min} that has the minimum MBR diagonal length (Lines 6–8). If the corresponding diagonal length does not exceed the distance threshold T_{dist}, we assign d to G_{min} and update partitioning $\mathcal{G}[w_i]$ (Lines 9–11); Otherwise, we create a new group G_{new} with the single document d and add G_{new} to $\mathcal{G}[w_i]$ (Lines 12–14).

[3] We do not index d_4 because it does not contain w_3.

Algorithm 2: GCOC

```
Input: Document partitioning list $\mathcal{G}$, New document $d$,
       Distance threshold $T_{dist}$
Output: Updated document partitioning $\mathcal{G}$
1  for each $w_i$ in $d.\psi$ do
2      Initialize $G_{min}$;
3      $diag_{min} \leftarrow +\infty$;
4      for each group $G$ in $\mathcal{G}[w_i]$ do
5          $G' \leftarrow G \cup \{d\}$;
6          if $diag(G') < diag_{min}$ then
7              $diag_{min} \leftarrow diag(G')$;
8              $G_{min} \leftarrow G$;
9      if $diag_{min} \le T_{dist}$ then
10         $G_{min}$.add($d$);
11         Update $G_{min}$ in $\mathcal{G}[w_i]$;
12     else
13         $G_{new} \leftarrow \{d\}$;
14         $\mathcal{G}[w_i]$.add($G_{new}$);
15 return;
```

Spatio-temporal popularity using network distance: Beyond Euclidean distance, the fast LTP score computation scheme is applicable to network distance. In particular, the network distance between two geographical points p_i and p_j is the minimum length of any sequence of road-network edges that connect p_i and p_j. Theorem 2 corroborates that the upper bound of the aggregated partial LTP score remains valid when using network distance.

Theorem 2 *Given a subscription s, a term w, a timestamp t, a document group G, and a new document d, the upper bound of $\mathsf{AP}_G(s, w, t)$ (i.e., $\mathsf{AP}_G(s, w, t).ub$) remains valid when $\mathsf{SF}(s, d, w)$ is computed based on network distance.*

Proof: *Recall that $minDist(s, G)$ in Equation 11 denotes the minimum Euclidean distance between s and G. Let $sd(s, d_i)$ be the network distance between s and d_i. We have: $\forall (d_i \in G)\ (minDist(s, G) \le sd(s, d_i))$. Based on Equation 11 we have:*

$$\sum_{d_i \in G} \frac{F(d_i, w) \cdot D^{-(t - d_i.t_c)}}{\alpha + sd(s, d_i)} \le \mathsf{AP}_G(s, w, t).ub$$

Using network distance, the left part of the above inequity is equal to $\mathsf{AP}_G(s, w, t)$ (cf. Equation 9). This completes the proof. □

3.3.3 Document and subscription updates

Document update: The document insertion scheme for subscription processing with the fast LTP score computation method is presented in Algorithm 2. To enable document group deletion, we evaluate the latest timestamp of each document group during subscription matching and remove outdated document groups from the inverted file. In particular, we remove as many outdated document groups as possible while guaranteeing that the accuracy of the LTP score is bounded by an error threshold Θ_{err}.

The detailed document group deletion scheme works as follows. After traversing posting list IL_w during subscription matching, we select the n document groups whose maximum possible errors of removal are the smallest. Here, n is the maximum value satisfying the following inequity:

$$\frac{Err_{acc}(w) \cdot D^{-(t-t_w)} + \sum_{i=1}^{n} Err_{max}(w, G_i)}{\mathsf{LTP}_{min}(w,t)} \leq \Theta_{err} \tag{14}$$

Here, $Err_{max}(w, G_i)$ denotes the maximum possible error incurred by deleting all documents in G_i, which is computed according to Equation 15, Θ_{err} is an error-rate threshold that guarantees that the error rate of LTP score incurred by deleting the n oldest postings does not exceed Θ_{err}, and $Err_{acc}(w)$ denotes the accumulated error at the timestamp t_w of the last deletion operation on IL_w. Each time we perform the document deletion operation, we update $Err_{acc}(w)$ and t_w.

$$Err_{max}(w, G) = \frac{F(w, G) \cdot D^{-(t-G.t_{max})}}{\alpha}, \tag{15}$$

where $F(w, G)$ indicates the sum of frequencies of w in G (i.e., $\sum_{d \in G} F(w, d)$), and $G.t_{max}$ denotes the latest timestamp of documents in G. Each time we complete traversing IL_w, we update $F(w, G)$, $G.t_{max}$, $Err_{acc}(w)$, and t_w.

Subscription updates: Since we have no index structure for SST subscriptions, each time when a new subscription is registered, we simply add the subscription to the subscription list. Similarly, if a user requests to de-register a subscription, we simply remove it from the subscription list.

3.4 SST Subscription Filtering based on Hierarchical Summarization

In our publish/subscribe setting, the number of SST subscriptions can be very large, making it very time consuming to evaluate SST subscriptions one by one. So we propose to evaluate groups of spatially similar subscriptions simultaneously. Specifically, we summarize SST subscriptions by recursively subdividing the underlying space into four quadrants (cells) [45] until the number of subscriptions in each cell is at most M, where M is a system parameter that controls the cell granularity. Each cell maintains a *summary label* for the SST subscriptions in the cell, which is the min-heap element of the subscriptions in the cell that has the lowest $w.p$ value. The summary label is used for generating cell filtering conditions to filter "unqualified" subscriptions.

Figure 7 illustrates the subscription filtering framework. The left figure presents the underlying space and existing SST subscriptions. Let $M = 2$. The underlying space is recursively divided into four equal-sized cells. Each cell contains at most 2 subscriptions[4]. After partitioning the space and subscriptions, we generate the summary label for each cell in a bottom-up fashion. Starting from the leaf cells, we select the top element of the min-heap for the subscription with the minimum current LTP score in a cell as its summary label. Assume that the current LTP score of the top element in $H[s_5]$ is less than that in $H[s_6]$ and $H[s_7]$ (i.e., $p_5 \cdot D^{-(t_{cur}-t_5)} < p_6 \cdot D^{-(t_{cur}-t_6)}$ and $p_5 \cdot D^{-(t_{cur}-t_5)} < p_7 \cdot D^{-(t_{cur}-t_7)}$). Let us consider cell c_{10} as an example. Two subscriptions, s_5 and s_6, fall into c_{10}. Let $\langle w_5, p_5, t_5 \rangle$ and $\langle w_6, p_6, t_6 \rangle$ be the top elements of $H[s_5]$ and $H[s_6]$, respectively. We select the one having the minimum current LTP score ($\langle w_5, p_5, t_5 \rangle$) as the summary label of c_{10}, which is denoted by $\ell_m(c_{10})$. Cell c_{11} contains only s_7, so we directly select $\langle w_7, p_7, t_7 \rangle$ as its summary label. Next, we generate the summary label for their parent cell c_7 that has four child cells in total, but where c_9 and c_{12} are empty. Because $p_5 \cdot D^{-(t_{cur}-t_5)} < p_7 \cdot D^{-(t_{cur}-t_7)}$), we pick $\langle w_5, p_5, t_5 \rangle$ as the summary label of c_7. We continue the summary label generation until the root cell c_0 is reached.

3.4.1 Subscription matching

Having the summary label, we are able to calculate the maximum LTP score between term w and subscriptions in cell c, which is denoted by $\mathsf{LTP}_{max}(c, w, t)$ and is computed as follows:

$$\mathsf{LTP}_{max}(c, w, t) = \sum_{d \in U} \frac{F(d, w) \cdot D^{-(t-d.t_c)}}{\alpha + minDist(c, d)} \tag{16}$$

where $minDist(c, d)$ represents the minimum distance between c and d. Note that $\mathsf{LTP}_{max}(c, w, t)$ can be computed efficiently by using Equations 11 and 13.

Theorem 3 *Given a term w and a cell c, w can be filtered by all* SST *subscriptions in cell c if:*

$\mathsf{LTP}_{max}(c, w, t_{cur}) \leq \ell_m(c).p \cdot D^{-(t_{cur}-\ell_m(c).t)}$

Proof: *The inequality suggests that $\forall s_i \in c$ ($\mathsf{LTP}(s_i, w, t)) \leq \ell_m(c).p \cdot D^{-(t_{cur}-\ell_m(c).t)}$). Since $\ell_m(c)$ has the lowest current LTP score among all subscriptions in c, according to Algorithm 1, Lines 7–8, we have that $\forall s_i \in c$ ($\mathsf{LTP}(s_i, w, t) \leq H[s_i].top().p$). This completes the proof.* □

Theorem 3 provides a subscription filtering condition that enables us to prune a set of unqualified SST subscription candidates that cannot include a term w from a new document as a top-k result. Next, we introduce the advanced matching algorithm that uses subscription filtering.

[4] Parameter M is set to 16–128 in experiments (cf. Section 5).

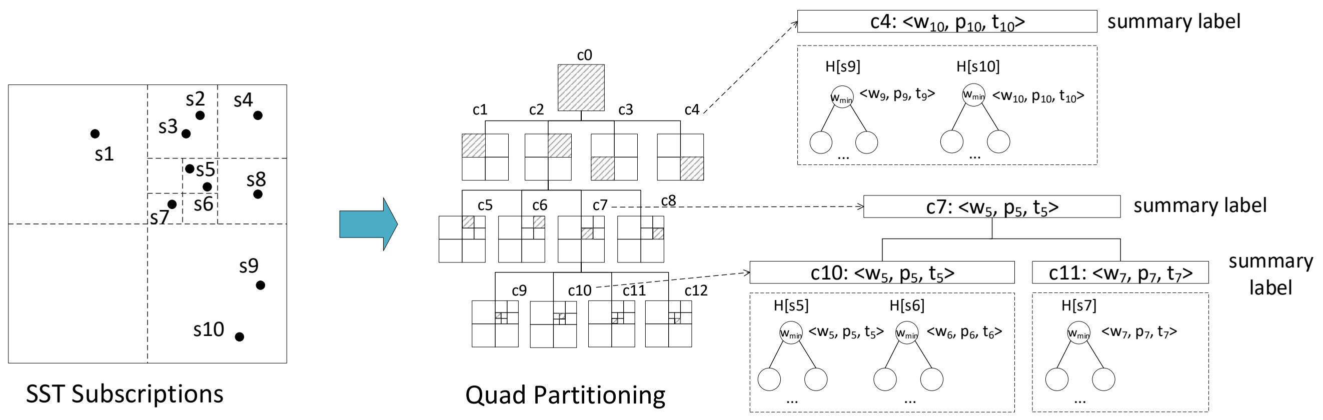


Fig. 7 SST subscription partitioning

Algorithm 3: HSMatching

```
Input: Root Cell $c_{root}$, new document $d$, inverted lists $IL$
Output: Updated inverted lists $IL$, updated min-heaps $H[s]$ of each subscription $s$
1 for each unique term $w_i$ in $d$ do
2     $IL[w_i]$.add($d$);
3     CellVisit($c_{root}$, $d$, $w_i$);
4 return;
```

Algorithm 4: CellVisit (Cell c, Document d, Term w)

```
Input: Cell $c$, summary label list $\ell_m$, min-heaps $H[s]$ of each subscription $s$, new document $d$
Output: Updated inverted lists $IL$, updated min-heaps $H[s]$ of each subscription $s$
1 if LTP$_{max}(c, w, t_{cur}) > \ell_m(c).p \cdot D^{-(t_{cur}-\ell_m(c).t)}$ then
2     for each child $c_i$ of $c$ do
3         CellVisit($c_i$,$d$,$w$);
4     if $c$ is a leaf cell then
5         for each $s \in c$ do
6             $P_{new} \leftarrow 0$;
7             for each posting $d_j$ in $IL[w_i]$ do
8                 $P_{new} \leftarrow P_{new} + F(d, w_i) \cdot D^{-(t-d_j.t_c)}/(\alpha + dist(s, d_j))$;
9             $w_t \leftarrow H[s].top()$;
10            $P_e \leftarrow w_t.p$;
11            $t_e \leftarrow w_t.t$;
12            $P_{cur} \leftarrow P_e \cdot D^{-(t_{cur}-t_e)}$;
13            if $H[s]$ contains $w_i$ then
14                $H[s]$.delete($w_i$);
15                $H[s]$.push($\langle w_i, P_{new}, t_{cur}\rangle$);
16            else if $P_{new} > P_{cur}$ then
17                $H[s]$.pop();
18                $H[s]$.push($\langle w_i, P_{new}, t_{cur}\rangle$);
19        while $\ell_m(c)$ is changed do
20            Update $\ell_m(c)$;
21            if $c$ is the root cell then
22                return;
23            $c \leftarrow c.parent$;
24 return;
```

Algorithm 3 presents the pseudo code of our subscription matching algorithm. We take the root cell c_{root}, new document d, and inverted lists as input. For each unique term w_i in d, we first add its posting into $IL[w_i]$ (Line 2). Next, we call recursive function CellVisit (cf. Algorithm 4) to evaluate each descendant cell (Line 3). Specifically, at first we check if subscriptions located cell c can be filtered based on Theorem 3. If so, we skip all subscriptions in c; Otherwise, we proceed to check c's four child cells (Lines 2–3). In particular, if c is a leaf cell we need to evaluate each subscription in c individually, which is the same as Algorithm 1, Lines 4–16. Finally, we update $\ell_m(c)$ from the leaf cell to the root cell (Lines 19–23).

Space complexity: The space complexity of the index structures used for processing SST subscriptions is $O(\psi_{avg} \cdot |\mathsf{D}| + (k+1) \cdot |S|)$, where ψ_{avg} denotes the average number of terms per document, D denotes the document collection, and S denotes the SST subscriptions.

3.4.2 Document and subscription updates

Document update: Our hierarchical summarization structure is designed to index subscriptions. It does not affect document insertion or deletion.

Subscription update: When a user registers a new SST subscription s, we apply Algorithm 5. Starting from the root cell, we traverse the cell that covers s in a top-down manner. When the number of subscriptions indexed by the cell exceeds M after indexing s (Line 2), we split the cell and index the subscriptions by its four child cells (Lines 3–4). After finding an appropriate cell to index s, we insert the min-heap of s ($H[s]$) into the cell and update its summary label (Line 5).

When a user de-registers an existing subscription s, we apply Algorithm 6. First, we locate the cell that indexes s (Line 1) and remove the min-heap of s ($H[s]$) from the cell (Line 2). Next, we repeatedly check whether the parent cell's

Algorithm 5: SubInsert

```
Input: Root Cell $c_{root}$, new subscription $s$, Cell capacity
       threshold $M$
Output: Updated hierarchical summarization structure
1 $c_{cur} \leftarrow c_{root}$;
2 while $|c_{cur}| \geq M$ do
3     Split $c_{cur}$;
4     $c_{cur} \leftarrow c_{cur}$'s child cell that covers $s$;
5 Insert $H[s]$ to $c_{cur}$ and update summary label;
6 return;
```

Algorithm 6: SubDelete

```
Input: Root Cell $c_{root}$, subscription $s$ to be deleted, Cell
       capacity threshold $M$
Output: Updated hierarchical summarization structure
1 $c_{cur} \leftarrow$ the cell that indexes $s$;
2 Remove $H[s]$ and update summary label;
3 $c_{cur} \leftarrow c_{cur}$'s parent cell;
4 while $|c_{cur}| \leq M$ do
5     Merge the child cells of $c_{cur}$;
6     $c_{cur} \leftarrow c_{cur}$'s parent cell;
7 return;
```

child cells can be merged until we reach the root cell. In particular, if the the number of subscriptions indexed by the descendants of a parent does not exceed M, we merge descendants (Lines 3–6).

4 RST Subscription Processing

This section presents baseline and our solution of processing a large number of RST subscriptions over a stream of spatio-temporal documents.

4.1 Baseline for Processing RST Subscriptions

The high-level idea of a straightforward method works as follows: For each term w in a new spatio-temporal document d we compute and update its popularity score in each RST subscription s; If it is greater than the popularity score of the current k-th term in s, w is regarded as a new result term and is used to update the current top-k term list for s. Note that the popularity score of each term in the top-k term list of s declines over time so we are required to re-compute them each time when a term from a new document arrives. The straightforward method is computationally prohibitive when the number of subscriptions is very large or the spatio-temporal documents arrive at a high rate. As a result, we need to develop a more efficient method to handling RST subscription over a stream of spatio-temporal documents.

It comes to our attention that an underlying idea of many publish/subscribe solutions (e.g., [4,55,63]) is to let similar subscription be organized together and evaluate them simultaneously, thus enhancing the performance of subscription processing. Nevertheless, it is challenging to achieve similar optimization goal for processing RST subscriptions. In order to avoid the re-evaluations of top-k term list maintained by each subscription and enable the computation sharing of similar subscriptions, we propose an approach including the following techniques to representing, indexing, and grouping RST subscriptions.

To maintain an exact top-k term list for each subscription in a real-time fashion, the straightforward method is to maintain a *temporal popularity table (TP table)* for each subscription s, which is basically a hashmap where the key is a term w from the documents falling in the subscription region ($s.r$), and the value is the current temporal popularity score ($\mathsf{TP}(s, w, t_{cur})$). The high-level idea of document processing is as follows. When a new document d_n is published, we first retrieve the RST subscriptions whose regions cover the location of d_n. After the pre-processing of d_n (i.e., de-composing the document into terms and removing stop words), for each retrieved subscription s we update its TP table. Finally, we update the top-k terms for each retrieved subscription. The time complexity for this result update algorithm is $O(k\ \log(k))$ for each matching between a term and a subscription.

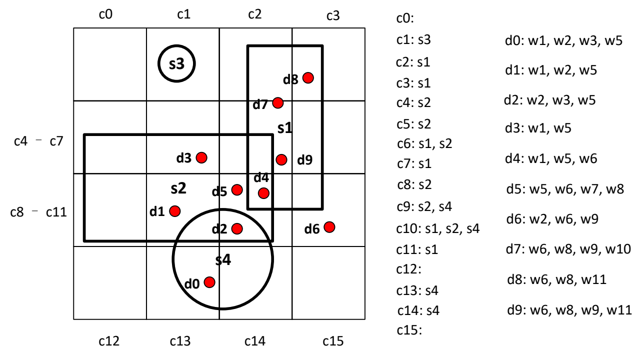


Fig. 8 RST subscriptions and documents

To facilitate the retrieval of RST subscriptions that cover a document, we can use grid indexing structure that partitions the spatial area into $m \times n$ congruent cells to index the spatial ranges of RST subscriptions, which is shown in Figure 8. For each cell we store the id of the subscriptions of which spatial ranges overlaps with the cell. For instance, we store subscriptions s_1, s_2, and s_4 under cell c_{10} because the spatial regions of s_1, s_2, and s_4 overlap with c_{10}. When a new document d_n arrives, subscriptions covering d_n are retrieved in a *filtering-and-refinement* fashion. For example, when d_4 arrives, subscriptions stored in c_{10} (i.e., s_1, s_2, and s_4) are retrieved because c_{10} covers d_4. Next, we proceed to check whether the subscriptions stored in c_{10} exactly cover

d_4. As a result, s_1 and s_2 cover d_4 while s_4 does not cover d_4.

After finding out the subscriptions of which regions covering d_n, we need to update the TP table and the top-k term list for each relevant subscription. Specifically, it works as follows: for each subscription, we maintain a hashmap to store the frequencies of all terms of documents falling in the subscription range, and we additionally keep a min-heap array of size k to record the TP scores of the top-k terms for each subscription.

Algorithm 7: ResultUpdate **(Subscription Index SI, Document d)**

```
1  $c \leftarrow$ the cell in $SI$ that covers $d.\rho$;
2  for each $s_i$ stored in $c$ do
3      if $s_i.r$ covers $d.\rho$ then
4          $M \leftarrow$ the HashMap of $s_i$;
5          $H \leftarrow$ the min-Heap of $s_i$;
6          for each term $w_j$ in $d$ do
7              Compute and update $\mathsf{TP}(s_i, w_j, t_{cur})$ in $M$
8              if key $w_j$ exists in $H$ then
9                  for each term $w_k$ in $H$ do
10                     Update $\mathsf{TP}(s_i, w_k, t_{cur})$ in $H$;
11                 MakeHeap($H$);
12             else
13                 $H.push(w_j)$;
14                 for each term $w_k$ in $H$ do
15                 Update $\mathsf{TP}(s_i, w_k, t_{cur})$ in $H$;
16                 MakeHeap($H$);
17                 $H.pop()$;
```

The pseudocode of updating the top-k terms maintained by each subscription is shown in Algorithm 7. The inputs are the subscription index SI and new document d. Initially, we initiate c to be the cell in SI covering the location of d, respectively (line 1). Next, for each subscription s_i of which region covers the location of d we need to update the top-k results for s_i indexed under c (lines 2–3). Note that M and H denote the hashmap and min-heap of s_i, respectively (lines 4–5). For each term w_j in d, we firstly compute and update $\mathsf{TP}(s_i, w_j, t_{cur})$ in M (line 7). Secondly, we need to update H. In particular, if w_j exists in H, we re-construct H by re-computing $\mathsf{TP}(s_i, w_k, t_{cur})$ for each w_k in H (lines 8–11). Else we insert w_j into H, re-construct H, and remove the top element in H (lines 12–17).

Subscription update: The subscription insertion and deletion operations of the baseline method are straightforward. Specifically, when a user registers an RST subscription s, we locate the cells that overlap with $s.r$ and store s in these cells. Similarly, when a user de-registers s, we remove s from the cells that overlap with $s.r$.

4.2 Overview of Technical Problems

Three limitations exist for the baseline algorithm. First, new documents are arriving in a streaming fashion and the temporal popularity score $\mathsf{TP}(s, w, t_{cur})$ of each term w in each subscription s is changing continuously as time elapses. Thus, for each subscription we need to re-construct its top-k term list (min-heap) when a new document arrives. Second, based on the publish/subscribe scenario, the number of RST subscriptions can be very large (i.e., 10M subscriptions is common for publish/subscribe systems). Hence, when a new spatio-temporal document d_n arrives we need to find the subset of RST subscriptions of which regions cover the location of d_n. Third, it is computationally expensive to maintain and continuously update a TP table for each subscription. We need to seek computation sharing solutions to enhance the efficiency of RST subscriptions processing when a new document arrives.

We proceed to present how to address the three technical problems respectively in detail. An underlying idea of many publish/subscribe solutions (e.g., [11, 20, 55]) is to group subscriptions such that they can be evaluated simultaneously for a new published item. Motivated by these systems, we also expect to design an approach to grouping RST subscriptions and their top-k term lists such that subscriptions in one group can be evaluated simultaneously, thus reducing the computation of subscription processing. Specifically, we define the concepts of *temporal popularity score index* and *subscription group*, which are used for filtering out the term from a new document that cannot be the top-k result for an individual RST subscription and a group of RST subscriptions, respectively. Based on the concepts we group and index the RST subscriptions by a set of non-overlapping cells from spatial index. Our framework of processing RST subscriptions consists of the following techniques.

(1) *Temporal popularity score index*, which is proposed for manipulating the temporal component of the TP score. Based on the temporal popularity score index, we develop an efficient algorithm for updating the top-k term list of each subscription. In particular, our algorithm is able to reduce the time complexity of updating the top-k result given a new term and a subscription from $O(k \log(k))$ to $O(\log(k))$. This technique will be presented in Section 4.3.

(2) *Subscription grouping technique*, which is used for: (1) sharing computations of updating TP scores of a term from different RST subscriptions; (2) checking whether a term from a new document can be a result of some subscription in a subscription group based on the group filtering condition. We define *subscription group* to help generate the group filtering condition for each set of subscriptions. Then we develop an efficient threshold-based online algorithm to generate the subscription groups. With the generated subscription groups, we develop an approach to handling RST

subscriptions in a group simultaneously for a term from a new document. This technique will be presented in Section 4.4.

Figure 9 illustrates our proposed architecture for processing RST subscriptions. Blue arrows denote the process of a subscription and green arrows denote the process of spatio-temporal documents. A user may both issue a subscription and generate a document.

When the system receives an RST subscription, the subscription is firstly initialized by traversing the *document index*, which is optional, then it is inserted into a subscription group that is most "similar" to the subscription. Subscription groups are indexed by a subscription index. For each subscription group, we maintain some summary information for enabling computation sharing when a new document arrives.

When a new document d_n arrives, we store it in a document index that indexes the spatial, text, and temporal information of each newly arrived document. Next, we consider each term w_i of d_n to be a "query" and we traverse the subscription index to retrieve the summary of each relevant subscription group. Then we evaluate whether the group can filter w_i. If a group cannot filter w_i, we proceed to evaluate each subscription s in the group to determine whether s can include w_i as their new top-k term.

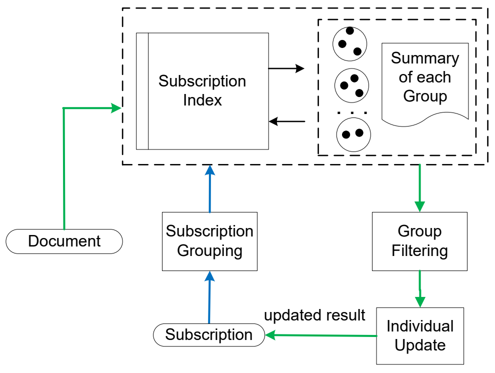


Fig. 9 Architecture for processing documents over RST subscriptions

4.3 Processing of a Single RST Subscription

In this section, we discuss the problem of how to monitor the updates of the top-k term list in an RST subscription that might be triggered by elapse of time and arrivals of new documents, respectively.

Lemma 4 *Let s be a* RST *subscription, w_i and w_j be two different terms. If* $\mathsf{TP}(s, w_i, t) > \mathsf{TP}(s, w_j, t)$ *and there is no document falling in $s.r$ during the time period $[t, t + \Delta t]$ ($\Delta t > 0$), we have* $\mathsf{TP}(s, w_i, t + \Delta t) > \mathsf{TP}(s, w_j, t + \Delta t)$.

Proof: According to Equation 3, if $\mathsf{TP}(s, w_i, t) > \mathsf{TP}(s, w_j, t)$*, we have:*

$$\sum_{d \in s.r} F(d, w_i) \cdot D^{-(t - d.t_c)} > \sum_{d \in s.r} F(d, w_j) \cdot D^{-(t - d.t_c)}.$$

So we have:

$$\sum_{d \in s.r} F(d, w_i) \cdot D^{-(t + \Delta t - d.t_c)} > \sum_{d \in s.r} F(d, w_j) \cdot D^{-(t + \Delta t - d.t_c)}.$$

As a result, we get $\mathsf{TP}(s, w_i, t + \Delta t) > \mathsf{TP}(s, w_j, t + \Delta t)$. □

Lemma 4 indicates that the temporal popularity of a term in a subscription has the following property: the relative ranking of two different terms w.r.t. a subscription is consistent over time. Hence, we do not need to update the top-k term list for each subscription over time. However, the difficulty here is that the absolute value of TP scores will decrease over time, which may affect the evaluation of whether a term from a new document can update the top-k term list in a subscription.

4.3.1 Top-k temporal popularity score index

To manipulate the update of time-varying TP scores, we develop a *top-k temporal popularity score index (TP index)* to store and maintain the top-k list, which consists of a hashtable for storing each term-score pair and its update timestamp, and a min-heap for indexing the current top-k terms with the highest TP scores.

Example 3 Figure 10 shows the TP index of RST subscription s. The TP index consists of two components: hashmap and min-heap. The key of the hashmap is the term (e.g., w_0) that is contained in the documents falling in the subscription region ($s.r$). The corresponding value in the hashmap is the TP score of w_0 at timestamp $t_u(w_0)$. In particular, $t_u(w_0)$ denotes the time when the TP score of w_0 in s was updated. The min-heap maintains top-k terms with the highest TP score. Assume that $k = 3$ and $\mathsf{TP}(s, w_9, t_u(w_9)) > \mathsf{TP}(s, w_5, t_u(w_5))$ and $\mathsf{TP}(s, w_5, t_u(w_5)) > \mathsf{TP}(s, w_3, t_u(w_3))$, the top element in the heap is term w_3 because w_3 has the minimum TP score among the scores of the three terms.

4.3.2 Update of TP index

Recall that when a new document d_n arrives, we firstly need to de-compose the spatio-temporal document into terms and remove stop words. Next we retrieve the RST subscriptions of which regions can cover the location of the new document. In the remaining parts of this paper, we use expression " document d_n matches subscription s" to denote that the

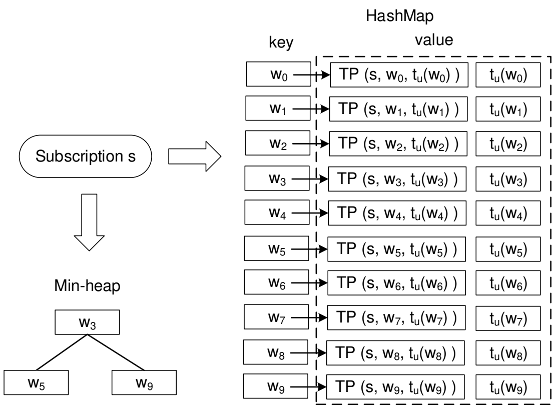


Fig. 10 TP index for subscription s

location of a new spatio-temporal document d_n falls in the spatial region of s. For each subscription that matches d_n we need to update its TP index to maintain the up-to-date top-k term list.

In this section, we present the problem that given a new spatio-temporal document d_n that matches a subscription s, how to update the TP index maintained by s. The high-level idea is as follows. For each term w_i in d_n we firstly retrieve the TP score of w_i in s from the hashmap and update it based on Lemma 5. Next, we update the min-heap. Specifically, if the TP score of w_i is grater than the TP score of the top element (term) in the Min-heap, we replace the top term with w_i.

Lemma 5 *Given* $\mathsf{TP}(s, w_i, t_u(w_i))$ *and* $t_u(w_i)$*, when a new document* d_n *containing* w_i *matches subscription* s *at current time* t_{cur}*, we have:*

$$\begin{aligned}&\mathsf{TP}(s, w_i, t_{cur})\\&= \mathsf{TP}(s, w_i, t_u(w_i)) \cdot D^{-(t_{cur}-t_u(w_i))} + F(d_n, w_i),\end{aligned} \tag{17}$$

where ϵ is an arbitrarily small value.

Proof: *Let d_1, d_2, d_3, ..., d_n be n documents ranked by their creation times ($d.t_c$). Assume that all of these n documents can match s. When d_{n-1} arrives, the TP score of w_i at the time of its arrival is:*

$$\sum_{j=1}^{n-1} F(d_j, w) \cdot D^{-(d_{n-1}.t_c - d_j.t_c)} = \mathsf{TP}(s, w_i, d_{n-1}.t_c).$$

When d_n arrives, the TP score of w_i at the time of its arrival is:

$$\sum_{i=j}^{n} F(d_j, w) \cdot D^{-(d_n.t_c - d_j.t_c)} = \mathsf{TP}(s, w_i, d_n.t_c).$$

When d_n arrives, $d_{n-1}.t_c$ is the time when the TP score of w_i in s was updated (i.e., $t_u(w_i)$), and $d_n.t_c$ is the current time t_{cur}. So we complete the proof. □

Note that the terms in min-heap are ranked based on the their current TP scores, which are computed by:

$$\mathsf{TP}(s, w_i, t_{cur}) = \mathsf{TP}(s, w_i, t_u(w_i)) \cdot D^{-(t_{cur}-t_u(w_i))}, \tag{18}$$

where $\mathsf{TP}(s, w_i, t_u(w_i))$ and $t_u(w_i)$ can be acquired from the hashmap.

Algorithm 8: TPUpdate **(Subscription s, Term w)**

```
1  $M \leftarrow$ the HashMap of $s$;
2  $H \leftarrow$ the min-Heap of $s$;
3  $val \leftarrow empty$;
4  if key $w$ exists in $M$ then
5      $val \leftarrow M.get(w)$;
6      Update $val.TP$ based on Equation 17;
7      $val.t_u \leftarrow currentTime$;
8      Update $val$ in $M$;
9  else
10     $val.TP \leftarrow F(d, w)$;
11     $val.t_u \leftarrow currentTime$;
12     Insert $w$–$val$ pair into $M$;
13 if key $w$ exists in $H$ then
14     Update $H$;
15 else
16     $w_t \leftarrow H.top()$;
17     if TP score of $w_t$ < TP score of $w$ then
18         Replace $w_t$ with $w$ in $H$;
19         Update $H$;
```

The pseudocode of TP index update is shown in Algorithm 8. The inputs are subscription s and a term w from a new document. Initially, we initiate M and H to be the hashmap and min-heap of s, respectively (lines 1–2). We retrieve its corresponding value in the hashmap M. Specifically, if M has key w, we get its values $val.TP$ and $val.t_u$, which represent $\mathsf{TP}(s, w_i, t_u(w))$ and $t_u(w)$, respectively (line 5). Next, we update the hashmap M by updating $val.TP$ based on Equation 17 and setting $val.t_u$ to be the current time (lines 6–8). If M does not have key w, we add a new w–val pair into M, where $val.TP$ is initialized as the frequency of w in d and $val.t_u$ is initialized as the current time (lines 9–12). After updating M, we need to update the min-heap H. Specifically, if w is in H, we update H based on the new TP score of w computed by Equation 18 (lines 13–14). Else we retrieve the top element (term) w_t in H and compare the current TP scores between w_t and w (lines 16–17). In particular, if the current TP score of w_t is smaller than that of w, we replace w_t with w in H and update H (lines 18–19).

4.4 Representing and Grouping RST Subscriptions

We propose an effective approach to representing and grouping RST subscriptions such that each group of subscriptions can be processed simultaneously for a term from a new spatio-temporal document. We first introduce the concept of RST Subscription Group (SG for short) to represent a group of similar RST subscriptions (Section 4.4.1). Based on the concept of SG, we present how to derive a *common TP index* for an SG, and show how to utilize the common TP index to share the computation of maintaining the top-k term result of each subscription (Section 4.4.2). Subsequently, we present our Threshold-based Online Group Generation (TOG) algorithm (Section 4.4.3)

4.4.1 RST *subscription group*

We propose the concept of RST Subscription Group (SG) to represent a group of RST subscriptions whose regions are spatially similar to each other. These subscriptions may have a much higher probability to share a common temporal temporal top-k term list, which can substantially reduce the term-subscription matching cost when a new document arrives.

Definition 6: RST *Subscription Group (SG).* Denoted by SG an RST subscription group is a set of RST subscriptions $\{s_0, s_1, s_2, ..., s_n\}$ where for any $s_i, s_j \in SG$, $s_i.r \cap s_j.r \neq \emptyset$. □

Based on Definition 6, there must exist a region that is shared by all RST subscriptions in a subscription group SG. Such region is defined as *Intersection region* of SG (i.e., $I(SG)$).

Definition 7: *Intersection Region.* Given a subscription group $SG = \{s_1, s_2 \cdots s_p\}$, it contains p subscriptions from s_1 to s_p. The intersection region for SG, denoted as $I(SG)$, is the largest region r satisfying the condition that for each $s_i \in SG$, $s_i.r \cap r = r$. □

According to Definition 7, the intersection region is the largest common region among all the regions associated with each subscription s_i. Thus, each subscription s satisfies that $s.r$ fully covers $I(SG)$. Similarly, we define the union bounding region in as follows.

Definition 8: *Union Bounding Region.* Given a subscription group $SG = \{s_1, s_2 \cdots s_p\}$, it contains p subscriptions from s_1 to s_p. The union bounding region for SG, denoted as $U(SG)$, is the smallest rectangular region r satisfying the condition that for each $s_i \in SG$, $s_i.r \cap r = s_i$. □

According to Definition 8, the union region is the smallest region that contains all the subscription regions. Thus for each subscription s_i, $U(SG)$ will contain $s_i.r$. With the definition of intersection region and union region, we define the compactness of a group, denoted as $C(SG)$, as $C(SG) = I(SG)/U(SG)$. $C(SG)$ evaluate the percentage of common regions as $I(SG)$ compared with the whole region, as $U(SG)$.

4.4.2 Common TP index and subscription group representation

To enable the computation sharing among the subscriptions in group SG, we build a common TP index for storing the TP score of terms from documents falling in the intersection region of SG (i.e., $I(SG)$). In particular, the structure of common TP index is the same as the structure of TP index introduced in Section 4.3.1 except that the common TP index does not have min-heap. Note that when a term from a new document falling in $I(SG)$ arrives, we just update the hashmap in the common TP index. The hashmaps in TP indices maintained by each $s_i \in SG$ are not required to be updated. However, we may still need to update the top-k term list (min-heap) in each $s_i \in SG$. Recall that the terms in the min-heap maintained by TP index of subscription s are ranked based on Equation 18, where the essential component (i.e., $\mathsf{TP}(s, w_i, t_u(w_i))$ and $t_u(w_i)$) can be acquired from the hashmap maintained by s. However, when we use common TP index to separately store the TP score of terms from documents falling in $I(SG)$, ranking the terms in min-heap based on Equation 18 is no longer valid because the TP score maintained by the hashmap of each $s_i \in SG$ only takes into account the documents published in the region $s.r \backslash I(SG)$.

We propose the *aggregated min-heap* to maintain the top-k terms for each $s_i \in SG$. Different from the original min-heap. The aggregated min-heap ranks its elements by aggregating (summing up) the TP scores maintained by hashmaps in common TP index and TP index of s_i respectively.

Recall that when a term w from a new document d_n arrives, for each subscription s of which region covers $d_n.\rho$ we need to retrieve the top element (term) w_t in the corresponding min-heap and compare the current TP scores between w_t and w (Algorithm 8). As shown in Figure 12, to enable group filtering we maintain two variables: (1) *minTop* that indicates the minimum value of TP scores; (2) *updateTime* that denotes the timestamp of the last update of *minTop*. In addition, we maintain a *max-hashmap* for SG where the key is term w and the value is the subscription s such that the TP score of w in s is the largest among all subscriptions in SG. We use $\mathsf{TP}(SG, w, t)$ to denote the TP score of w maintained by the common TP index of SG, use $s_{max}(w, SG)$ to denote the subscription such that the TP score of w in s is the largest among all subscriptions in SG, and use $SG.currentMinTop$ to denote the value of $minTop$ at current time, which can be derived by aggregating $minTop$ and $updateTime$ based on Equation 18. Then we have Theorem 4, which is regarded as the group filtering condition.

Theorem 4 *Let $SG = \{s_0, s_1, ...s_{p-1}\}$ be a subscription group with p* RST *subscriptions, w be a term from a new document d_n and $I(SG)$ covers $d_n.\rho$. w cannot be a top-k*

result of any $s_i \in G$ if the following condition is satisfied:

$$\mathsf{TP}(SG, w, t_{cur}) + \mathsf{TP}(s_{max}(w, SG), w, t_{cur}) \leq SG.currentMinTop \quad (19)$$

Proof: *Let $\mathsf{TP}_{-}(s_i, w, t_{cur})$ be the current TP score maintained by s_i and si $\in SG$. If the top-k term list of s_i can be updated by w, then we have:*

$$\mathsf{TP}(SG, w, t_{cur}) + \mathsf{TP}_{-}(s_i, w, t_{cur}) > SG.currentMinTop. \quad (20)$$

Because $\mathsf{TP}(s_{max}(w, SG), w, t_{cur}) \geq \mathsf{TP}_{-}(s_i, w, t_{cur})$, we complete the proof. □

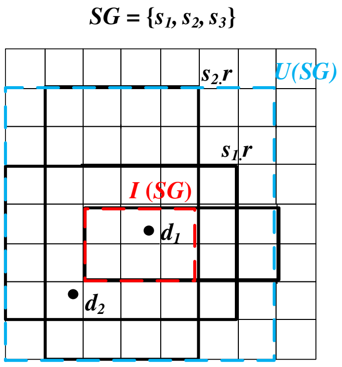


Fig. 11 Subscription group SG

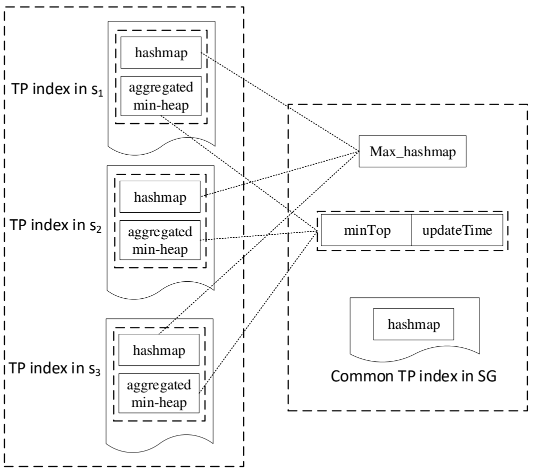


Fig. 12 Representation of subscription group

Example 4 Let SG be a subscription group containing RST subscriptions s_1, s_2, and s_3. The spatial regions of s_1, s_2, and s_3 (i.e., $s_1.r$, $s_2.r$, and $s_3.r$, respectively) are depicted by dark solid rectangles in Figure 11. The intersection region ($I(SG)$) and the union bounding region ($U(SG)$) of SG are depicted by the red dash rectangle and blue dash rectangle, respectively. When a new document d_1 arrives, we find that $d_1.\rho$ falls in $I(SG)$. In this case, we only need to update the hashmap in the common TP index. When another document d_2 arrives, we find that $d_2.\rho$ falls outside $I(SG)$ but it falls in $s_1.r$ and $s_2.r$. Here we need to update the hashmaps in the TP indices maintained by s_1 and s_2 separately.

Algorithm 9: TOG (**Subscription** s, **GroupSet** $\mathcal{G}$, **Threshold** θ)

```
1:  $create \leftarrow$ true;
2:  $MinChange \leftarrow \infty$;
3:  for each group $SG_i \in \mathcal{G}$ do
4:      $SG' = s \cup SG_i$;
5:      if $s.r \cap I(SG_i) \neq \emptyset$ and $C(SG') > \theta$ then
6:          $create \leftarrow$ false;
7:          if $MinChange > C(SG') - C(SG_i)$ then
8:              $MinChange \leftarrow C(SG') - C(SG_i)$;
9:              $insert \leftarrow i$;
10:         end if
11:     end if
12: end for
13: if $create = true$ then
14:     create a new group $SG$ including $s$;
15:     insert $SG$ into $\mathcal{G}$;
16: else
17:     insert $s$ into the group $SG_{insert}$;
18: end if
```

Space complexity: The space complexity of the subscription group index is $O((\mathsf{T}_{avg} + \mathsf{G}_{avg} + k) \cdot |S|)$, where T_{avg} denotes the average number of terms for documents located in an RST subscription, G_{avg} denotes the average number of grid cells for indexing each subscription, and $|S|$ is the number of subscriptions.

4.4.3 Online Generation of Subscription Groups

We proceed to present our algorithm for generating subscription groups over the RST subscriptions. Based on the publish/subscribe scenario, the arrival rate of subscriptions is much lower than the arrival rate of published items. Nevertheless, we still need to regard the RST subscriptions as a data stream rather than a static dataset. As a consequence, we proposed a threshold-based online algorithm for generating subscription groups over a stream of RST subscriptions. In particular, we focus on the problem that given a new subscription s and a set of subscription groups, how to assign a group that is most "similar" to s for enhancing the computation sharing in processing documents over a set of subscriptions.

Algorithm 9 presents the pseudo code of our thesholdbased online group generation (TOG) algorithm. S include all the existing subscriptions and has formed all the existing groups $SG_1, SG_2 \cdots$. s is the new subscription. *create* keeps whether s can be inserted into the existing group and is set to be true initially (line 1). $MinChange$ keeps the changes after inserting s into the group and set to be ∞ (line 2). Then for each existing group $SG_i \in \mathcal{G}$, a new group SG' will be constructed by inserting s into SG_i (line 4). After that, the intersection regions between $s.r$ and $I(SG_i)$ as well as $C(SG')$ will be computed because a subscription s can be inserted into SG_i only if $s.r \cap I(SG_i) \neq \emptyset$ and the compactness for the new group SG' is larger than the threshold θ (line 5). If the condition is satisfied, the subscription s can be inserted into at least one existing group, thus $create$ will be set as false (line 6). In the next steps, the algorithm will keep the information for the inserted group. The $MinChange$ will keep the minimum value among all the valid inserted groups. If $MinChange > C(SG') - C(SG_i)$ (line 7), it indicates that inserting s into SG_i will make the minimum change smaller, then $MinChangte$ will be updated as $C(SG') - C(SG_i)$ (line 8) and $insert$ will store the number of the group as i (line 9). After checking all the groups, if $create$ is true, then we cannot insert s into any existing groups. In this scenario, a new group SG that contains a single element s will be generated (line 14) and inserted into $\mathcal{G}$ (line 15). Otherwise, we insert s into the existing group SG_{insert} (line 17).

Subscription update: When a user registers a new RST subscription s, we apply Algorithm 9 to insert s into an appropriate group. When a user de-registers s, we mark s as "de-registered" while leaving it in its group. We remove a group SG if all subscriptions in the group are marked as "de-registered."

4.5 Associating Subscription Group with Spatial Index

Recall that each RST subscription s needs to be associated with a set of non-overlapping spatial cells that can cover the whole area of $s.r$ (Section 4.1). Likewise, we need to associate each subscription group SG with cells of a spatial index. To choose such a set of cells for a better performance, we propose an effective method to associate SG onto a set of Quad-tree cells.

Admittedly, we may use other spatial indices (e.g., Grid index, R-tree based indices) for the purpose. Specifically, R-tree based spatial index is applicable for our problem, but it is a less suitable structure in comparison with the Quad-tree. The reason is that the structure of the R-tree will be greatly dependent on the distribution of subscription locations. It will incur much additional cost for maintaining the MBRs (i.e., split and merge operations) when new queries arrive. Such shortcoming also exists for the R+-tree and the R*-tree. As for the grid index, we have to determine the granularity of the grid. Because that different subscriptions may have diverging spatial regions, it is difficult to determine a unified granularity that is suitable for all subscriptions. As a consequence we choose the Quad-tree based indexing structure to index RST subscription groups.

Given an RST subscription group SG, when a new spatio-temporal document d_n arrives, if $U(SG)$ does not cover $d_n.\rho$ then we do not need to evaluate any $s_i \in SG$. Hence, we use $U(SG)$ to represent the spatial region of SG.

We propose a heuristic method for associating a subscription group onto the Quad-tree cells:

Step 1: Starting from the root cell c_r of the Quad-tree, we check the number of c_r's children that covers $U(SG)$. If it equals to 1, we invoke Step 2 with the c_r's child who covers $U(SG)$ as the input; Else we return c_r.

Step 2: Given an input cell c, we check the number of c's children that covers $U(SG)$. If it equals to 1, we invoke Step 2 with the c's child who covers $U(SG)$ as the input; Else we return c.

Such recursive procedure terminates when there is no single cell that can cover the whole region of $U(SG)$.

4.6 Algorithm for RST Subscription Matching

Now we are ready to present the algorithm for processing a new document over a set of subscription groups. Recall that the RST subscriptions are grouped based on TOG algorithm (Section 4.4.3) and each subscription group SG is associated with a set of non-overlapping cells that cover $U(SG)$. Hence, when a new spatio-temporal document d_n arrives, we only traverse the subscription groups in the cells that cover $d_n.\rho$.

Algorithm 10: DocumentProcess(**Document** d, **Index** SI)

```
c ← $SI.root$;
while cell c is not empty do
    for each subscription group SG in c do
        if $U(SG)$ covers $d.\rho$ then
            if $I(SG)$ covers $d.\rho$ then
                for each term $w_i \in d.\psi$ do
                    Update common TP index of $SG$;
                    if $SG$ cannot filter out $w_i$ based on Theorem 1 then
                        for each $s_j \in SG$ do
                            TPUpdate($s_j, w_i$);
                        Update max-hashmap, $minTop$, and $updateTime$;
            else
                for each $s_j \in SG$ do
                    if $d.\rho$ covers $s_j.r$ then
                        for each term $w_i \in d.\psi$ do
                            TPUpdate($s_j, w_i$);
                        Update $max_hashmap$, $minTop$, and $updateTime$;
    c ← c's child node that contains $d.\rho$;
```

Algorithm 10 presents the pseudo code of the algorithm for processing a new document over a set of subscription groups. The input of this algorithm is the new document d and the existing groups of subscription indexed by the Quad-tree SI. At the beginning, we set c as the root node of SI (line 1). We traverse the Quad-tree cells that cover the location of d in a depth-first manner (line 2). Next, we evaluate each subscription group SG stored under c (line 3). In particular, if $U(SG)$ covers the location of d, it means that there may exist a subscription in SG such that the subscription region can cover the location of d (line 4). Then we proceed to check whether $I(SG)$ covers the location of d (line 5). If so, we firstly update the common TP index, and then for each term w_i in d we check whether the subscriptions in SG can filter out w_i on the basis of Theorem 4. If not, we need to each each s_i in SG separately. In particular, we call algorithm TPUpate. After that, we update the summary information maintained by SG (lines 8–11). If the location of d does not fall in $I(SG)$, we also need to evaluate each s_i in SG separately. Specifically, we firstly check whether the location of d falls in the region of s_i. Next we call algorithm TPUpate for each term in d. After that, we update the summary information maintained by SG (lines 13–17).

5 Experimental Study

We report on experiments with real data that offer insight into important properties of the developed algorithms. Four methods are evaluated in our experiments:

(1) Baseline – Baseline algorithm for processing SST subscriptions, presented in Section 3.1;

(2) TS – Subscription matching with tailored result update, introduced in Section 3.2;

(3) FTS – TS optimized by fast computation of the LTP score, presented in Section 3.3;

(4) HSFTS – FTS optimized by hierarchical summarization based subscription filtering, covered in Section 3.4.

5.1 Datasets and Subscription Generation

Our experiments are conducted on two datasets: FSD and TWD[5]. FSD is a real-life dataset collected from Foursquare, which contains worldwide POIs with geographical information. The dataset TWD is a larger dataset that comprises geo-tagged tweets with locations. After pre-processing, the average numbers of terms per item are 5.6 and 9.3 for FSD and TWD, respectively, and the total sizes of FSD and TWD are 83 MB and 3.2 GB, respectively.

SST subscription generation: To generate SST subscriptions from FSD, we randomly select 1 million POIs. The selected POI locations are regarded as subscription locations (i.e., $s.\rho$ in Definition 2). The number of results k is set to be 10 as default. We also conduct an experiment that studies the effect of varying parameter k. Similarly, we randomly select a particular number of tweet locations as subscription locations for TWD.

RST subscription generation: The RST subscriptions are generated as follows. For FSD, the location of each POI is used as the center of an RST subscription region, and we generate a rectangular or circular region of a pre-defined area with that center. The resulting region is used as the region of an RST subscription. For TWD, we randomly select a particular number of tweets, and the center of each subscription region is the same as the location of each tweet. The shape of a region r is determined as follows: Let the area of the region r be a. The probability that r is a rectangle equals 50%, and the probability that r is a circle equals 50%, too. If r is a rectangle, the longer side of r is parallel to the x-axis (50% probability) or the y-axis (50% probability). The length/width ratio is uniformly distributed from in the range from 1 to 2.

Additionally, for experiments on FSD, we regard each tweet in TWD as a spatio-temporal document, and we regard each region generated from the POIs as the region of an RST subscription. For experiments on TWD, each tweet in TWD is considered to be a spatio-temporal document on TWD. Each region generated from the randomly selected tweets is considered to be the region of an RST subscription.

5.2 Experimental Results for SST Subscriptions

We present the experiment settings and results related to the processing of SST subscriptions. Parameter ranges and default values are presented in Table 3.

	FSD	TWD
number of result terms (k)	1–30 default: 10	1–500 default: 10
preference parameter (α)	$d_{max} \times 10^{-6}$ – $d_{max} \times 10^{-2}$ default: $d_{max} \times 10^{-4}$	$d_{max} \times 10^{-5}$ – $d_{max} \times 10^{-1}$ default: $d_{max} \times 10^{-3}$
distance threshold (T_{dist})	100km–500km default: 300km	50km–200km default: 100km
decaying scale	0.1–0.9 default: 0.5	0.1–0.9 default: 0.5
granularity parameter (M)	16–128 default: 32	16–128 default: 32
number of subscriptions	1M	10M–40M default:10M

Table 3 Parameter ranges and default values

[5] http://lisi.io

Four methods are implemented in Java: Baseline[6], TS, FTS, and HSFTS. Experiments in Euclidean space are conducted on server with an Intel Xeon(R) Gold 5120 CPU @2.20GHz and 64GB RAM. Experiments in network space are conducted on a server with an Intel Xeon(R) Gold 5120 CPU @2.20GHz and 256GB RAM.

5.2.1 Time effect

In this set of experiments, each method is run for 6,000 seconds (which is called the simulation duration, denoted by Δt_{sim}) on both FSD and TWD. We set the decaying scale $D^{-\Delta t_{sim}}$ to 0.5. The default values of other parameters are presented in Table 2. For each method, we continuously publish spatio-temporal documents (e.g., geo-tagged tweets). To enable the least efficient method to be able to process the data stream, we publish 2 spatio-temporal documents per second on FSD and 3 spatio-temporal documents every 10 seconds on TWD. In addition, we register a new SST subscription and de-register an existing subscription in the subscription pool every 5 seconds. Each method is initialized with 1M and 10M SST subscriptions, for FSD and TWD, respectively, and we warm up each method with 30,000 spatio-temporal documents.

We report the average runtime for processing a document (i.e., the sum of the runtimes for processing each term in the document, including subscription matching, result update, and document group update) and the average runtime for inserting a subscription during each period of 1,000 seconds. Figure 13 shows that HSFTS exhibits the best performance on both datasets and that FTS substantially outperforms TS in document processing. In particular, from Figure 13(a) we find that the performance of TS decreases as time elapses. In contrast, for FTS and TSFTS the performance remains consistent. The reason is that TS computes an LTP score by visiting every posting (document) in an inverted list and that the lengths of inverted lists grow as time elapses. However, FTS and HSFTS compute LTP scores by means of document groups. The number of document groups is much smaller than the number of postings. Figure 13(b) does not show a similar trend because the document arrival rate for TWD is much lower than that for FSD. We also find that HSFTS performs significantly better than FTS, which confirms the effectiveness of our subscription filtering technique.

Figure 14 shows the performance of SST subscription update (i.e., insertion and deletion) for each method. TS and FTS perform better than HSFTS. The reason is that HSFTS requires each new subscription to be indexed for generating a hierarchical summarization. However, in general publish/subscribe scenarios, the frequency of subscription update is normally much lower than that of item arrivals, and thus the subscription update cost will be substantially smaller than that of document processing. Furthermore, when considering Figure 13 and Figure 14 together, we find that on dataset FSD, HSFTS exhibits similar runtimes in terms of document processing and subscription update ($\sim$6ms). In contrast, for FTS and TS, the runtimes of document processing are $\sim$20ms and $\sim$70ms, respectively, which are much larger than the subscription update time of HSFTS. As a result, when accounting for both document processing and subscription update, HSFTS still performs much better than TS and FTS. In the rest of experiments, we focus on evaluating document processing.

5.2.2 Effect of k

This experiment evaluates the performance of the four algorithms when varying the number of result terms maintained by each subscription. As shown in Figure 15, the document processing runtime of Baseline exhibits a conspicuous increasing trend when we increase the value of k. In contrast, the other three algorithms (TS, FTS, and HSFTS) show smaller runtime increases. The reason is that Baseline needs to re-order the min-heap maintained by each updated SST subscription after processing each document. In contrast, the other three algorithms adopt the tailored result update technique, which reduces the time complexity of result updates from $O(|S| \cdot k \cdot \log k)$ to $O(|S| \cdot \log k)$.

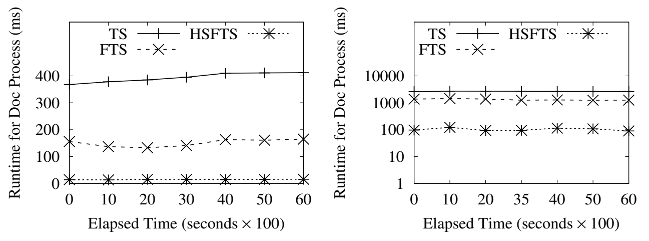
5.2.3 Effect of distance preference parameter

We proceed to evaluate the effect of distance preference parameter α. Specifically, we vary α from $d_{max} \times 10^{-6}$ to $d_{max} \times 10^{-2}$ and from $d_{max} \times 10^{-5}$ to $d_{max} \times 10^{-1}$ on FSD and TWD, respectively, where d_{max} indicates the maximum distance in the underlying space. Note that a lower value of α denotes higher weight of the spatial proximity. Figure 16 shows that the performance of TS is relatively unaffected by α. In contrast, both FTS and HSFTS exhibit improved efficiency when we decrease the value of α. This result can be explained by the fact that the LTP score computation and subscription filtering techniques are based on the spatial attribute of documents and subscriptions, respectively. Hence, increasing the weight of the spatial proximity in the LTP score can enhance the grouping and filtering effects.

5.2.4 Effect of distance threshold

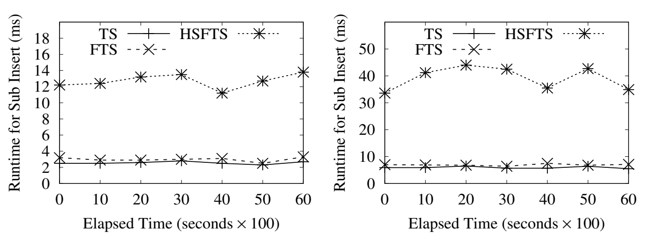
This experiment evaluates the effect of distance threshold T_{dist}. From Figure 17 we learn that FTS and HSFTS both produce the best performance when we set T_{dist} to 300 km

[6] The performance discrepancy between Baseline and TS is negligible when k is small. Thus, we only report the result of Baseline when varying k.



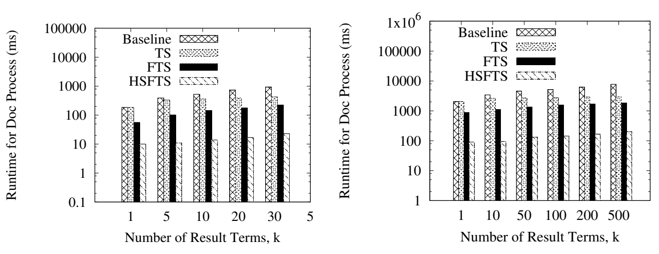
(a) FSD (b) TWD

Fig. 13 Effect of elapsed time regarding document processing



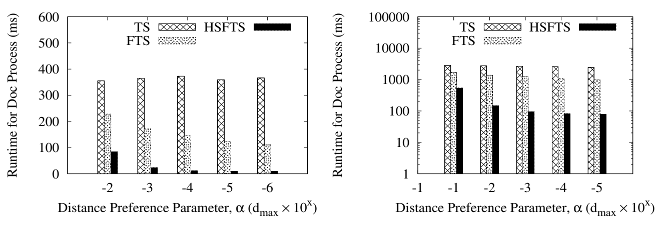
(a) FSD (b) TWD

Fig. 14 Effect of elapsed time regarding subscription update



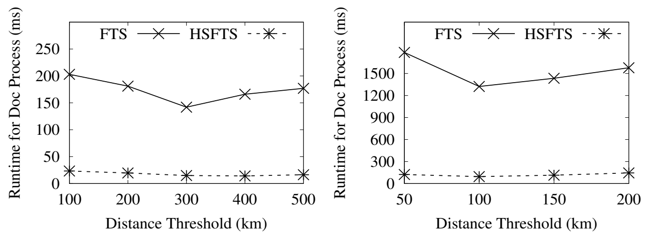
(a) FSD (b) TWD

Fig. 15 Effect of the number of results, k



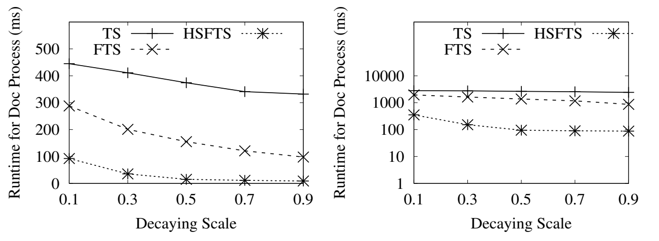
(a) FSD (b) TWD

Fig. 16 Effect of distance preference parameter, α



(a) FSD (b) TWD

Fig. 17 Effect of distance threshold, T_{dist}



(a) FSD (b) TWD

Fig. 18 Effect of decaying scale

and 100 km on FSD and TWD, respectively. In particular, when T_{dist} is small, the number of groups associated with each inverted list can be very large, which increases the cost of computing the LTP score. On the other hand, if we increase the value of T_{dist}, we will have fewer document groups. Nevertheless, the average MBR diagonal length of each group is be increased, which may loosen the bound on the LTP score.

5.2.5 *Effect of decaying Scale*

From Figure 18 , we can see that the runtime for document processing decreases as we increase the decaying scale. The reason is that a higher decaying scale value decreases the number of matched subscriptions. In particular, the runtime of TS only decreases slightly as we increase the decaying scale. Further, the runtimes of FTS and HSFTS exhibit moderate decreasing trends when we increase decaying scale. The reason is that fewer matched subscriptions result in more subscriptions being filtered in advance.

5.2.6 *Effect of granularity parameter*

This experiment evaluates the performance when varying the maximum number of subscriptions in a leaf cell (M). As indicated in Figure 19(a), HSFTS achieves the best performance when M is set at 32.

5.2.7 *Scalability*

Finally, we evaluate the scalability aspect in terms of document processing efficiency and memory cost. From Figure 20(a) we can see that when the number of subscriptions is increased to 40 million, HSFTS remains capable of processing a spatio-temporal document within 160 milliseconds, which is much more than an order of magnitude faster than TS. From Figure 20(d), we observe that the memory cost of HSFTS is just slightly higher than that of TS (10%–15%). Figure 20(b) reports the maximum stream arrival rate that can be sustained by each method when the number of subscriptions varies from 10M to 40M. Given 10M subscriptions, HSFTS can support a data stream with an arrival rate of 630 docs/min. In contrast, TS and FTS can only sustain streams with rates of 22 docs/min and 43 docs/min, respec-

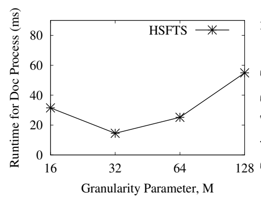
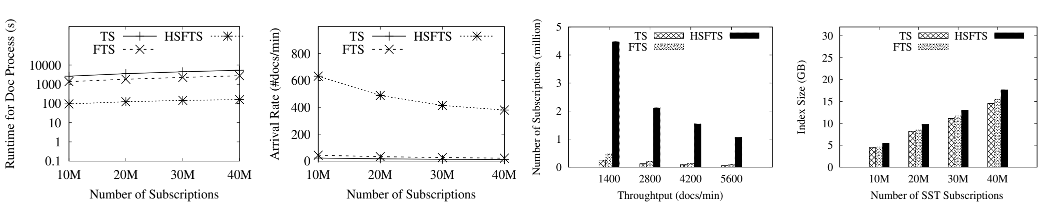


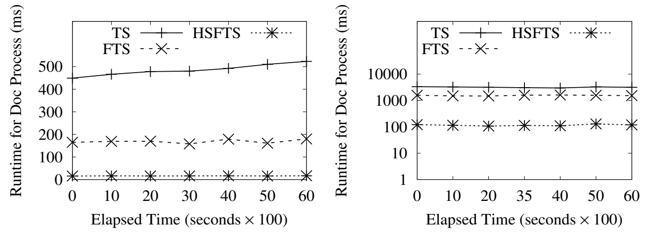
Fig. 19 Granularity



(a) runtime (b) arrival rate (c) subscription cardinality (d) memory cost

Fig. 20 Scalability

tively. When we increase the number of subscriptions to 40M, HSFTS can still sustain a stream with a rate of 380 docs/min, which is 35x and 18x better than for TS and FTS, respectively. Figure 20(b) reports the maximum number of subscriptions that can be handled by each method when the stream rate varies from 1,400 to 5,600 docs/min. In the real-world scenario, the average arrival rate of geo-tagged tweets is $\sim$2,800 docs/min [57]. Additionally, Figure 20(c) reports that HSFTS is able to handle 2.1 million SST subscriptions over geo-textual data streams with a real-world arrival rate. In contrast, FTS and TS can only support 0.202 million and 0.116 million subscriptions, respectively.



(a) FSD (network) (b) TWD (network)

Fig. 21 Effect of elapsed time regarding document processing on road networks

5.2.8 Time effect of road networks

This set of experiments evaluates the efficiency of document matching on road networks. Specifically, we map locations to a U.S. road network graph[7], which consists of 175,812 nodes and 179,179 edges. The distance between a subscription s and a document d (i.e., $dist(s, d)$ in Equation 2) is the shortest network distance between s and d. We pre-compute the all-pair shortest path distances using Dijkstra's algorithm [21], which has complexity $O(|V| \cdot |E| + |V|^2 \cdot \log |V|)$, where $|V|$ and $|E|$ denote the total numbers of nodes and edges, respectively, and we store the pre-computed results in memory (126.5 GB). For each method, we continuously publish spatio-temporal documents (e.g., geo-tagged tweets). Similarly, to enable the least efficient method to be able to process the data stream, we publish 3 spatio-temporal documents every 2 seconds in the case of FSD and 2 spatio-temporal documents every 10 seconds in the case of TWD. FSD and TWD are initialized with 1M and 10M SST subscriptions respectively, and we warm up each method with 30,000 spatio-temporal documents. Figure 21 shows that when we use network distance, the time costs for all methods are slightly higher than the corresponding time costs when using Euclidean distance (cf. Figure 13). The results show that the fast LTP score computation method remains capable of computing a good approximation of the exact LTP scores when we use network distance.

[7] http://www.cs.utah.edu/ lifeifei/research/tpq/

5.3 Experimental Results for Processing RST Subscriptions

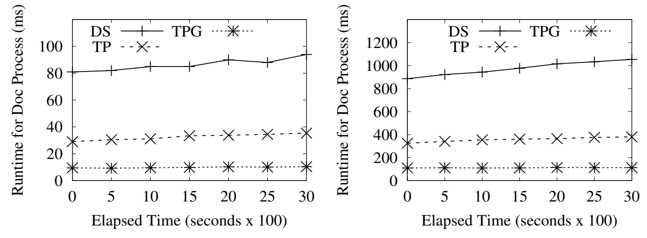
	FSD	TWD
RST subscription region size	1km^2–10,000km^2 default: 100km^2	1km^2–10,000km^2 default: 100km^2
number of result terms (k)	1–30 default: 10	1–30 default: 10
group generation parameter (θ)	0.1–0.9 default: 0.5	0.1–0.9 default: 0.5
decaying scale	0.1–0.9 default: 0.5	0.1–0.9 default: 0.5
number of subscriptions	1M	10M–40M default: 10M

Table 4 Parameter ranges and default values

We implemented the following three methods in Java on a PC with Intel(R) Core(TM) i7-3770 @3.40GHz and 16GB RAM. Note that experiments on scalability are conducted on server with an Intel Xeon(R) Gold 5120 CPU @2.20GHz and 64GB RAM. (1) DS – Direct processing of RST subscriptions (Section 4.1); (2) TP – Processing subscriptions with TP index (Section 4.3.1); (3) TPG – Processing subscriptions with TP index and subscription grouping mechanism (Section 4.6);

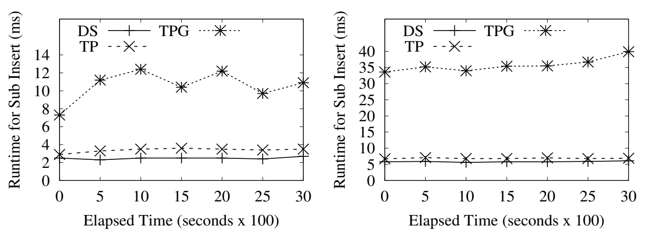
5.3.1 Time effect

In this set of experiments, each method runs for 3,000 seconds (which is simulation duration, denoted by Δt_{sim}) on both FSD and TWD. We set the decaying scale $D^{-\Delta t_{sim}}$ at 0.5. For each method, we continuously publish spatio-temporal documents (i.e., geo-tagged tweets). We publish



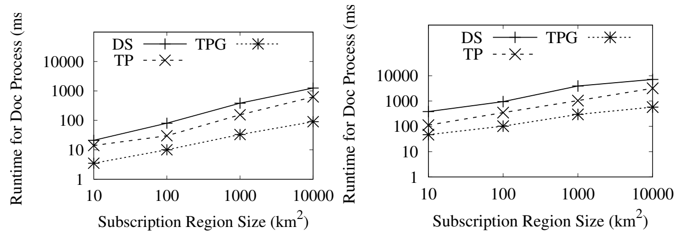
(a) FSD (b) TWD

Fig. 22 Effect of time for document processing



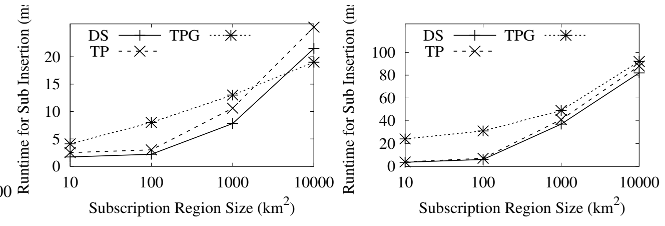
(a) FSD (b) TWD

Fig. 23 Effect of time for subscription insertion



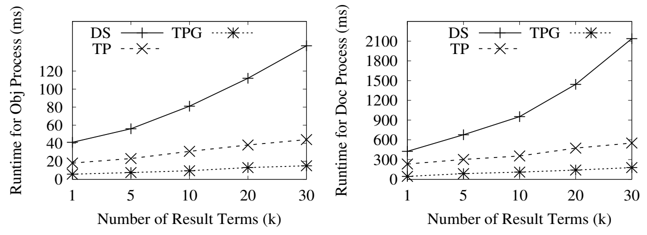
(a) FSD (b) TWD

Fig. 24 Effect of subscription region size for document processing



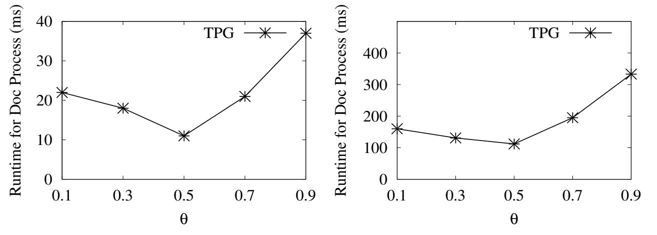
(a) FSD (b) TWD

Fig. 25 Effect of subscription region size for subscription insertion



(a) FSD (b) TWD

Fig. 26 Effect of result cardinality (k)



(a) FSD (b) TWD

Fig. 27 Effect of group generation parameter (θ)

the i+1-st document as soon as the processing of the i-th document is completed. In addition, during each second we register a new RST subscription and de-register an existing subscription each second. At the beginning, FSD and TWD are initialized with 1M and 10M RST subscriptions, respectively, and we warm up each method with 10,000 spatio-temporal documents. In other words, we start issuing RST subscription and reporting the runtime performance when the processing of the 10,001-st document begins.

We report the average runtime for processing a document (i.e., the sum of runtime for processing each term in the document) and the average runtime for inserting a subscription during each period of 500 seconds. Figure 22 shows that TPG exhibits the best performance. In particular, TPG substantially outperforms TP and greatly outperforms DS in document processing. TP performs much better than DS. TPG is able to improve the runtime performance of DS by approximately an order of magnitude on both datasets. The reasons could be explained as follows.

For DS, we need to re-construct the minheap maintained by each subscription of which region covers a new document, which is very time-consuming. While for TP, minheap re-construction can be avoided because of its TP index maintained by each subscription. Consequently, TP performs substantially better than DS. However, compared with TPG, TP does not group the similar subscriptions, and thus it is impossible to let spatially similar subscriptions share computations in document processing. On the contrary, TPG enables group evaluation and group filtering while updating the top-k term list of each subscription, which make a big contribution to improving the efficiency.

Figure 23 shows the performance of subscription insertion for each method. Since TPG triggers TOG algorithm (Section 4.4.3) when a new subscription arrives, we need evaluate the “similarity” between the region of the new subscription and each subscription group. Therefore, the runtime costs of subscription insertion for TPG is higher than TP and DS. In particular, DS performs slightly better than TP in terms of subscription insertion. This can be explained by the additional cost for initializing TP index. However, the time cost of subscription insertion is greatly lower than the time cost of document processing by comparing the runtime in Figure 22. Hence, by accounting for both document processing and subscription insertion, TPG still reduces the runtime of DS by $\sim$90%.

5.3.2 *Size of subscription region*

We proceed to evaluate the effect of the region size of each RST subscription. Figure 24 shows that all the methods present an increasing trend for the runtime of object processing as we increase the area of subscription region. This is because the average area of subscription region is proportional to the average number of subscriptions stored under each spatial cell, which will lead to an increase in the number of subscriptions that we need to evaluate when a new document arrives. We also observe that TPG is able to improve on the runtime of DS by an order of magnitude. Figure 25 demonstrates the runtime of subscription insertion as we vary the subscription region area. Similar to Figure 24, all the methods present an increasing trend for the runtime of subscription insertion as we increase the region area.

5.3.3 *Number of result terms*

This experiment evaluates the performance w.r.t. parameter "k". Figure 26 shows that for TPG and TP the runtime for document processing slightly increases as we increase the result cardinality k. The reason is that higher value of k is likely to induce the lower TP score of the top element (term) in the min-heap maintained by each subscription. Thus, the average number of subscriptions that have their min-heap updated will increase, which may lower the performance of document processing. In addition, we find that the increasing trend of DS is much more significant than that of TPG and TP. Such contrast can be explained by the min-heap reconstruction of DS while evaluating each subscription.

5.3.4 *Effect of θ*

In this experiment, we investigate the effect of the group generation parameter θ. We observe similar trends on both datasets. In particular, $\theta = 0.5$ yields the best performance. If θ increases, it is more likely for a new subscription to generate a new group. Thus, the number of subscription groups will mount up. Let us consider an extreme case where $\theta = 1$. In this case, each group contains one subscription, which will be degenerated to method TP. On the other hand, if θ is too small, the spatial similarity between subscriptions within a group will decrease, which may harm the efficiency of document processing.

5.3.5 *Grid granularity*

Finally, we evaluate the effect of grid granularity for methods using grid index to store RST subscriptions Figure 28(a) demonstrates the document processing performances of DS and TP when we vary the grid cell size from 1km$\times$1km to 50km$\times$50km. We observe that the grid granularity has little effect on the document processing performances of DS and TP.

5.3.6 *Scalability*

We evaluate the scalability of each method regarding the efficiency and the memory cost. Figures 29(a) and 29(d) show that the runtime for document processing and the memory cost increase linearly for all methods as we increase the number of subscriptions. Further, Figure 29(c) shows that TPG can support 1.83 million RST subscriptions over geo-textual data streams with real-world arrival rate (i.e., 2,800 docs/min). In contrast, TP and DS can support 481 thousand and 192 thousand subscriptions, respectively. In addition, based on Figure 29(d) we can see that the memory cost of TPG is only slightly higher than that of DS, which underlines that the TP index and the subscription groups are not space consuming.

5.4 Case Study

To exemplify the subscription results, we grab the real-time top-30 most popular terms returned by SST subscription s_1, and RST subscriptions s_2 and s_3. Specifically, the location of s_1 (the red dot) is the center of Chicago, IL, and the regions of s_2 (the purple rectangle) and s_3 (the green circle) cover the urban area of Chicago. The top-30 result terms are visualized by Word Clouds, where the font size of a term is proportional to its popularity score. Both result sets contain terms related to trending news or topics around Chicago (e.g., *drug*, *park*). On the other hand, we observe that the result terms returned by s_2 are inclined to be more "localized" compared with the result returned by s_1. As indicated by s_2, terms *wrigley* (a chewing gum company based in Chicago) and *notre* (a Chicago shop selling sneakers and apparel) are regarded as popular terms in Chicago, while neither of these two terms are returned by s_1. As for the result of s_1, we can see more terms regarding general trending topics (e.g., *NFL*, *football*, *election*). Such difference in results can be explained by the fact that an RST subscription exclusively considers documents located in its subscription region and that it filters out any other documents located outside the region. In contrast, an SST subscription takes all documents from data streams as its input while still favoring the terms from documents near its subscription location.

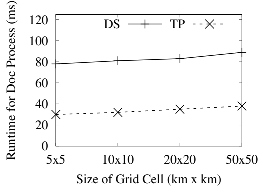
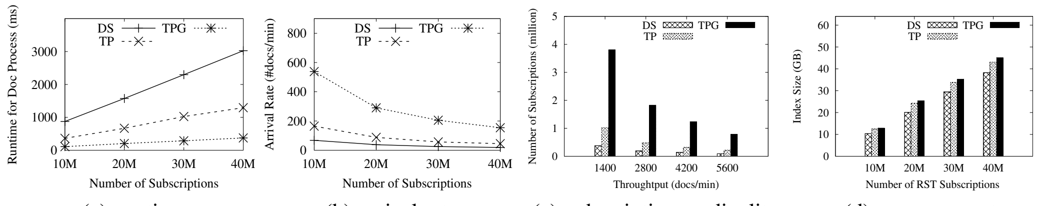


Fig. 28 Grid size



(a) runtime (b) arrival rate (c) subscription cardinality (d) memory cost

Fig. 29 Scalability

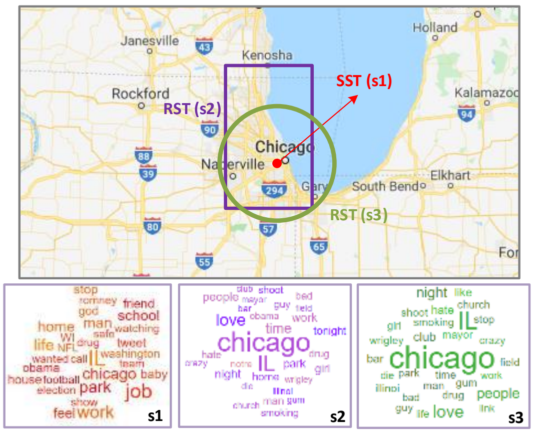


Fig. 30 Result snapshots for SST and RST subscriptions

6 Related Work

6.1 Location-based Publish/Subscribe

A number of publish/subscribe systems are developed for spatio-temporal documents. Given a new spatio-temporal document d, the spatial matching condition for subscriptions proposed in some literature [10, 61, 67] is that d falls in the subscription region (if d is a point) or d overlaps the query region (if d is a region) [33,67]. While for others, a score that measures the spatial proximity between the subscription location and the location of a new spatio-temporal document d [11, 12, 28, 29], or a score that measures the spatial overlap between a subscription and a region of spatio-temporal documents [13, 18, 68], is computed. Recently, Mahmood et al. [38] target the problem of indexing continuous spatio-textual queries by capturing the variations in the frequencies of keywords across different spatial regions, which significantly enhance the efficiency of processing spatio-textual queries.

However, the published items defined in these location-based publish/subscribe systems are spatio-temporal documents, which cannot be applied to our publish/subscribe systems where the published items are terms.

6.2 Content-based Top-k Publish/Subscribe

Top-k publish/subscribe scores new messages for each subscription and delivers new messages that rank among the top-k published messages for the subscription.

The problem of top-k publish/subscribe has been extensively studied in previous literature, where inverted file is used as the subscription index and the classic information retrieval methods are adapted for the ranking. Specifically, the subscriptions studied by some proposals (e.g., [7,26,44]) maintain the messages that are ranked among the k most relevant ones published within a time period defined by a sliding window. When a relevant message of a subscription expires, it is replaced by the most relevant message in the sliding window through a re-evaluation algorithm. To avoid re-evaluation, Shraer et al. [55] integrate a decaying factor into the ranking score, which gradually decays as time elapses. Because the decaying rate is the same for all published messages, it is guaranteed that older messages retire from the top-k set of a subscription only when new messages with higher scores arrive.

6.3 Spatio-temporal Search

Several studies focus on finding top-k most frequent terms over static or dynamic sets of spatio-temporal documents. Mokbel et al. [43] and Magdy et al. [36] offer a comprehensive tutorial and survey, respectively, on this topic. Given a set of spatio-temporal documents, In particular, Skovsgaard et al. [56], Ahmed et al. [3], and Hong et al. [58] offer means of finding top-k most frequent terms in documents that belong to a specified region and timespan. Magdy et al. [37] propose a scalable and efficient query system, GeoTrend, that, given a stream of spatio-temporal documents, is able to find the top-k most frequent terms in documents that belong to an arbitrary spatial region and the timespan from the current time and T time units into the past. GeoTrend is arguably the state-of-the-art system for processing trending one-time spatio-temporal term queries. It is capable of sustaining high data arrival rates while maintaining a low query latency. Jonathan et al. [31] study the problem of finding top-k trending terms within an arbitrary subset of documents selected based on their attributes. Additionally, Abdelhaq et al. [1,2] focus on extracting local keywords from a Twitter stream by identifying local keywords and estimating the central location of each keyword. Wang et al. [59] identify local top-k maximal frequent keyword co-occurrence patterns over streams of geo-tagged tweets. Other studies on spatio-temporal search include counting-based term queries [16, 64], spatial keyword search over geo-textual data streams [15,66], route planning [8,14,24,47,52–54,62], and trajectory search [46,48–51,71].

The major difference between our proposal and these studies is that we process a large number of continuous queries, i.e., we keep the results of a large population of "standing" queries up-to-date with respect to incoming data from when the queries are registered until when they are deregistered; in contrast, the other studies concern the efficient processing of "one-time" queries, i.e., standard queries that are processed once as they arrive.

7 Conclusion and Future Work

We study the problem of maintaining the up-to-date locally trending terms for a large number of term subscriptions over a stream of spatio-temporal documents. Specifically, we define two types of term-based subscriptions: SST subscription and RST subscription. Term frequency, spatial proximity, and term freshness are taken into consideration for publishing and subscribing. We develop efficient solutions to maintain real-time results for a large number of SST and RST subscriptions. Our experimental results suggest that our proposal is able to achieve reductions of the processing time by 70%–95% and 60%– 90% for processing SST subscriptions and RST subscriptions, respectively, compared with baselines developed based on existing techniques.

However, this study develops two separate index structures to process SST and RST subscriptions, respectively. We also observe that existing proposals on location-based term publish/subscribe either focus on region-based subscriptions or focus on rank-order subscriptions. In the future, we propose to develop a generic publish/subscribe framework for processing both region-based subscriptions (e.g., RST) and rank-order subscriptions (e.g., SST) in an efficient manner.

Acknowledgements This work was supported by the National Natural Science Foundation of China (61932004, 61922054, 61872235, 61729202, 61832017, U1636210), the National Key Research and Development Program of China (2018YFC1504504, 2016YFB0700502), and Hong Kong RGC grant 12201018.

References

1. H. Abdelhaq and M. Gertz. On the locality of keywords in twitter streams. In *IWGS*, pages 12–20, 2014.
2. H. Abdelhaq, M. Gertz, and A. Armiti. Efficient online extraction of keywords for localized events in twitter. *GeoInformatica*, 21(2):365–388, 2017.
3. P. Ahmed, M. Hasan, A. Kashyap, V. Hristidis, and V. J. Tsotras. Efficient computation of top-k frequent terms over spatio-temporal ranges. In *SIGMOD*, pages 1227–1241, 2017.
4. M. Altinel and M. J. Franklin. Efficient filtering of xml documents for selective dissemination of information. In *VLDB*, pages 53–64, 2000.
5. G. Amati, G. Amodeo, and C. Gaibisso. Survival analysis for freshness in microblogging search. In *CIKM*, pages 2483–2486. ACM, 2012.
6. P. G. Anick. Using terminological feedback for web search refinement: a log-based study. In *SIGIR*, pages 88–95, 2003.
7. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.
8. X. Cao, L. Chen, G. Cong, and X. Xiao. Keyword-aware optimal route search. *PVLDB*, 5(11):1136–1147, 2012.
9. L. Chen and G. Cong. Diversity-aware top-k publish/subscribe for text stream. In *SIGMOD*, pages 347–362, 2015.
10. L. Chen, G. Cong, and X. Cao. An efficient query indexing mechanism for filtering geo-textual data. In *SIGMOD*, pages 749–760, 2013.
11. L. Chen, G. Cong, X. Cao, and K. Tan. Temporal spatial-keyword top-k publish/subscribe. In *ICDE*, pages 255–266, 2015.
12. L. Chen and S. Shang. Approximate spatio-temporal top-k publish/subscribe. *World Wide Web*, 22(5):2153–2175, 2019.
13. L. Chen and S. Shang. Region-based message exploration over spatio-temporal data streams. In *AAAI*, pages 873–880, 2019.
14. L. Chen, S. Shang, C. S. Jensen, B. Yao, Z. Zhang, and L. Shao. Effective and efficient reuse of past travel behavior for route recommendation. In *KDD*, pages 488–498, 2019.
15. L. Chen, S. Shang, C. Yang, and J. Li. Spatial keyword search: a survey. *GeoInformatica*, 24(1):85–106, 2020.
16. L. Chen, S. Shang, B. Yao, and K. Zheng. Spatio-temporal top-k term search over sliding window. *World Wide Web*, 22(5):1953–1970, 2019.
17. L. Chen, S. Shang, Z. Zhang, X. Cao, C. S. Jensen, and P. Kalnis. Location-aware top-k term publish/subscribe. In *ICDE*, pages 749–760, 2018.
18. L. Chen, S. Shang, K. Zheng, and P. Kalnis. Cluster-based subscription matching for geo-textual data streams. In *ICDE*, pages 890–901, 2019.
19. Z. Chen, G. Cong, Z. Zhang, T. Z. Fuz, and L. Chen. Distributed publish/subscribe query processing on the spatio-textual data stream. In *ICDE*, pages 1095–1106, 2017.
20. Y. Diao, P. M. Fischer, M. J. Franklin, and R. To. Yfilter: Efficient and scalable filtering of XML documents. In *ICDE*, pages 341–342, 2002.
21. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
22. M. Efron and G. Golovchinsky. Estimation methods for ranking recent information. In *SIGIR*, pages 495–504. ACM, 2011.
23. A. Farzindar and W. Khreich. A survey of techniques for event detection in twitter. *Computational Intelligence*, 31(1):132–164, 2015.
24. D. Guo, Y. Zhu, W. Xu, S. Shang, and Z. Ding. How to find appropriate automobile exhibition halls: Towards a personalized recommendation service for auto show. *Neurocomputing*, 213:95–101, 2016.
25. L. Guo, D. Zhang, G. Li, K. Tan, and Z. Bao. Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In *SIGMOD*, pages 843–857, 2015.
26. P. Haghani, S. Michel, and K. Aberer. The gist of everything new: Personalized top-k processing over web 2.0 streams. In *CIKM*, pages 489–498, 2010.

27. Q. He, K. Chang, E. Lim, and J. Zhang. Bursty feature representation for clustering text streams. In *SDM*, pages 491–496, 2007.
28. H. Hu, Y. Liu, G. Li, J. Feng, and K. Tan. A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In *ICDE*, pages 711–722, 2015.
29. J. Hu, R. Cheng, D. Wu, and B. Jin. Efficient top-k subscription matching for location-aware publish/subscribe. In *SSTD*, pages 333–351, 2015.
30. M. Hu, S. Liu, F. Wei, Y. Wu, J. T. Stasko, and K. Ma. Breaking news on twitter. In *CHI Conference on Human Factors in Computing Systems, CHI '12, Austin, TX, USA - May 05 - 10, 2012*, pages 2751–2754, 2012.
31. C. Jonathan, A. Magdy, M. F. Mokbel, and A. Jonathan. GARNET: A holistic system approach for trending queries in microblogs. In *ICDE*, pages 1251–1262, 2016.
32. H. Kwak, C. Lee, H. Park, and S. B. Moon. What is twitter, a social network or a news media? In *WWW*, pages 591–600, 2010.
33. G. Li, Y. Wang, T. Wang, and J. Feng. Location-aware publish/subscribe. In *KDD*, pages 802–810, 2013.
34. X. Li and W. B. Croft. Time-based language models. In *CIKM*, pages 469–475. ACM, 2003.
35. H. Liang, Y. Xu, D. Tjondronegoro, and P. Christen. Time-aware topic recommendation based on micro-blogs. In *CIKM*, pages 1657–1661, 2012.
36. A. Magdy, L. Abdelhafeez, Y. Kang, E. Ong, and M. F. Mokbel. Microblogs data management: a survey. *VLDB J.*, pages 1–40, 2019.
37. A. Magdy, A. M. Aly, M. F. Mokbel, S. Elnikety, Y. He, S. Nath, and W. G. Aref. Geotrend: spatial trending queries on real-time microblogs. In *SIGSPATIAL*, pages 7:1–7:10, 2016.
38. 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.
39. G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357, 2002.
40. M. Mathioudakis, N. Bansal, and N. Koudas. Identifying, attributing and describing spatial bursts. *PVLDB*, 3(1):1091–1102, 2010.
41. M. Mathioudakis and N. Koudas. Twittermonitor: trend detection over the twitter stream. In *SIGMOD*, pages 1155–1158, 2010.
42. A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, pages 398–412, 2005.
43. M. F. Mokbel and A. Magdy. Microblogs data management systems: Querying, analysis, and visualization. In *SIGMOD*, pages 2219–2222, 2016.
44. K. Pripuzic, I. P. Zarko, and K. Aberer. Top-k/w publish/subscribe: Finding k most relevant publications in sliding time window w. In *DEBS*, pages 127–138, 2008.
45. H. Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, 1984.
46. S. Shang, L. Chen, C. S. Jensen, J. Wen, and P. Kalnis. Searching trajectories by regions of interest. *IEEE Trans. Knowl. Data Eng.*, 29(7):1549–1562, 2017.
47. S. Shang, L. Chen, Z. Wei, C. S. Jensen, J. Wen, and P. Kalnis. Collective travel planning in spatial networks. *IEEE Trans. Knowl. Data Eng.*, 28(5):1132–1146, 2016.
48. S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Trajectory similarity join in spatial networks. *PVLDB*, 10(11):1178–1189, 2017.
49. S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Parallel trajectory similarity joins in spatial networks. *VLDB J.*, 27(3):395–420, 2018.
50. S. Shang, L. Chen, K. Zheng, C. S. Jensen, Z. Wei, and P. Kalnis. Parallel trajectory-to-location join. *IEEE Trans. Knowl. Data Eng.*, 31(6):1194–1207, 2019.
51. S. Shang, R. Ding, K. Zheng, C. S. Jensen, P. Kalnis, and X. Zhou. Personalized trajectory matching in spatial networks. *VLDB J.*, 23(3):449–468, 2014.
52. S. Shang, J. Liu, K. Zheng, H. Lu, T. B. Pedersen, and J. Wen. Planning unobstructed paths in traffic-aware spatial networks. *GeoInformatica*, 19(4):723–746, 2015.
53. S. Shang, H. Lu, T. B. Pedersen, and X. Xie. Finding traffic-aware fastest paths in spatial networks. In *SSTD*, pages 128–145, 2013.
54. S. Shang, H. Lu, T. B. Pedersen, and X. Xie. Modeling of traffic-aware travel time in spatial networks. In *MDM*, pages 247–250, 2013.
55. A. Shraer, M. Gurevich, M. Fontoura, and V. Josifovski. Top-k publish-subscribe for social annotation of news. *PVLDB*, 6(6):385–396, 2013.
56. A. Skovsgaard, D. Sidlauskas, and C. S. Jensen. Scalable top-k spatio-temporal term querying. In *ICDE*, pages 148–159, 2014.
57. L. Sloan and J. Morgan. Who tweets with their location? understanding the relationship between demographic characteristics and the use of geoservices and geotagging on twitter. *PLoS ONE*, 10(11), 2015.
58. L. H. Van and A. Takasu. Parallelizing top-k frequent spatiotemporal terms computation on key-value stores. In *SIGSPATIAL*, pages 476–479, 2018.
59. X. Wang, Y. Zhang, W. Zhang, and X. Lin. Efficient identification of local keyword patterns in microblogging platforms. *IEEE Trans. Knowl. Data Eng.*, 28(10):2621–2634, 2016.
60. 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.
61. 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.
62. Y. Wang, J. Li, Y. Zhong, S. Zhu, D. Guo, and S. Shang. Discovery of accessible locations using region-based geo-social data. *World Wide Web*, 22(3):929–944, 2019.
63. X. Xiong, M. F. Mokbel, and W. G. Aref. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *ICDE*, pages 643–654, 2005.
64. Y. Xu, L. Chen, B. Yao, S. Shang, S. Zhu, K. Zheng, and F. Li. Location-based top-k term querying over sliding window. In *WISE*, pages 299–314, 2017.
65. Y. Xu, K. Wang, B. Zhang, and Z. Chen. Privacy-enhancing personalized web search. In *WWW*, pages 591–600, 2007.
66. C. Yang, L. Chen, S. Shang, F. Zhu, L. Liu, and L. Shao. Toward efficient navigation of massive-scale geo-textual streams. In *IJCAI*, pages 4838–4845, 2019.
67. M. Yu, G. Li, and J. Feng. A cost-based method for location-aware publish/subscribe services. In *CIKM*, pages 693–702, 2015.
68. M. Yu, G. Li, T. Wang, J. Feng, and Z. Gong. Efficient filtering algorithms for location-aware publish/subscribe. *IEEE Trans. Knowl. Data Eng.*, 27(4):950–963, 2015.
69. K. Zhao, L. Chen, and G. Cong. Topic exploration in spatio-temporal document collections. In *SIGMOD*, pages 985–998, 2016.
70. K. Zhao, Y. Liu, Q. Yuan, L. Chen, Z. Chen, and G. Cong. Towards personalized maps: Mining user preferences from geo-textual data. *PVLDB*, 9(13):1545–1548, 2016.
71. Y. Zhao, S. Shang, Y. Wang, B. Zheng, Q. V. H. Nguyen, and K. Zheng. REST: A reference-based framework for spatio-temporal trajectory compression. In *KDD*, pages 2797–2806, 2018.