World Wide Web (2019) 22:1953–1970
https://doi.org/10.1007/s11280-018-0606-x

Spatio-temporal top-*k* term search over sliding window

Lisi Chen[1] · Shuo Shang[2] · Bin Yao[3] · Kai Zheng[4]

Received: 25 April 2018 / Revised: 16 May 2018 / Accepted: 25 May 2018 /
Published online: 18 June 2018

Abstract In part due to the proliferation of GPS-equipped mobile devices, massive volumes of geo-tagged streaming text messages are becoming available on social media. It is of great interest to discover most frequent nearby terms from such tremendous stream data. In this paper, we present novel indexing, updating, and query processing techniques that are capable of discovering top-*k* most frequent nearby terms over a sliding window. Specifically, given a query location and a set of geo-tagged messages within a sliding window, we study the problem of searching for the top-*k* terms by considering term frequency, spatial proximity, and term freshness. We develop a novel and efficient mechanism to solve the problem, including a quad-tree based indexing structure, indexing update technique, and a best-first based searching algorithm. An empirical study is conducted to show that our proposed techniques are efficient and fit for users' requirements through varying a number of parameters.

This article belongs to the Topical Collection: *Special Issue on Web Information Systems Engineering 2017*
Guest Editors: Lu Chen and Yunjun Gao

✉ Shuo Shang
jedi.shang@gmail.com

Lisi Chen
lisi@uow.edu.au

Bin Yao
yaobin@cs.sjtu.edu.cn

Kai Zheng
zhengkai@uestc.edu.cn

[1] University of Wollongong, Wollongong, Australia

[2] King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

[3] Shanghai Jiao Tong University, Shanghai, China

[4] University of Electronic Science and Technology of China, Chengdu, China

Keywords Top-k · Term · Spatial · Temporal

1 Introduction

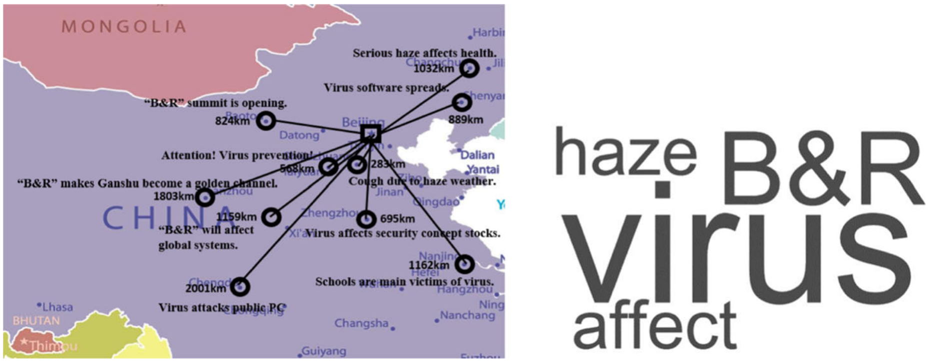
With the proliferation of social media, cloud storage, and location-based services, the amount of messages containing both text and geographical information (e.g., geo-tagged tweets) are skyrocketing. Such messages, which can be modeled as geo-textual data streams, often offer first-hand information for a variety of local events of different types and scale, including breaking news stories in an area, urban disasters, local business promotions, and trending opinions of public concerns in a city.

Data streams from location-based social media bear the following natures: (1) *bursty nature* - messages regarding a particular topic can be quickly buried deep in the stream if the user is not fast enough to discover it [28]; (2) *local-intended nature* - users from different locations may post messages related to diverging topics [62]. With thousands of messages being generated from location-based social media each second, it is of great importance to maintain a summary of what occupies minds of users.

To address the problem, existing proposal [47] aims at finding the top-k locally popular terms in content within a user-specified spatio-temporal region. However, in most cases it is difficult for a user to specify a rectangular region on the spatial domain. Instead, a user may prefer a rank-ordered list of terms by taking both term frequency and location proximity into consideration.

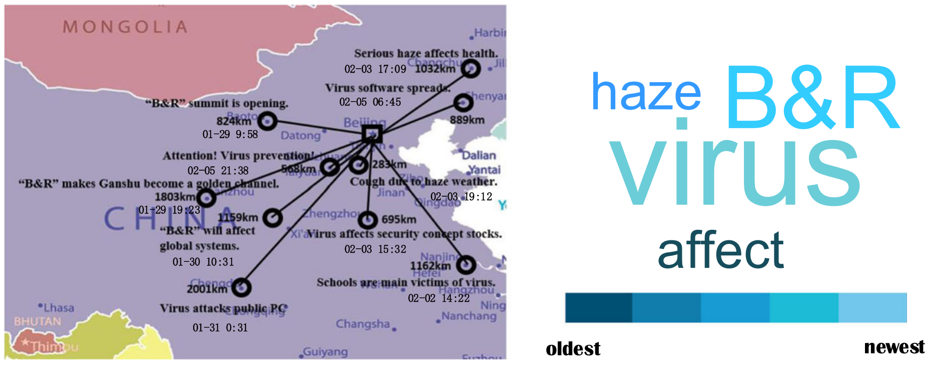
Based on the user requirements, we consider two kinds of top-k term query, $\underline{\text{L}}$ocation-based Top-$\underline{k}$ $\underline{\text{T}}$erm $\underline{\text{Q}}$uery (LkTQ) and $\underline{\text{S}}$patio-$\underline{\text{T}}$emporal Top-$\underline{k}$ $\underline{\text{T}}$erm $\underline{\text{Q}}$uery (STkTQ). In particular, the LkTQ returns top-k locally frequent terms by taking into account both location proximity and term frequency for geo-textual data over a sliding window, and the STkTQ returns top-k locally trending terms by taking into account location proximity, term frequency, and term freshness for geo-textual data over a sliding window.

Figure 1 provides a toy example of LkTQ . Let us consider 10 geo-tagged tweets located on the map of China. The point with square label indicates the query location. The points with circle labels are geo-textual messages. For each geo-textual message, we present its textual information and corresponding distances to the query point. The results of the LkTQ



(a) Messages and distances (b) Tag Cloud

Figure 1 Example of LkTQ



(a) Messages, distances, and timestamps (b) Tag Cloud

Figure 2 Example of ST*k*TQ

are the k most locally popular terms based on a location-aware frequency score, which are shown in Figure 1b. The score of a term is computed by a linear combination of the term frequency and location proximities between the query and the messages containing the term. Figure 2 presents an example of ST*k*TQ. Besides textual and spatial information, ST*k*TQ takes temporal information into consideration (i.e., the timestamp of each message). For each geo-textual message, we present its textual information, distance to the query point, and timestamp. The results of the ST*k*TQ are the k most locally trending terms based on a Spatio-temporal frequency score, which are shown in Figure 2b. The score of a term is computed by a combination of the term frequency, location proximities, and freshness of messages that contain the term.

A straightforward approach for answering an L*k*TQ or an ST*k*TQ is to evaluate all terms of messages within the current sliding window. Specifically, for each of such terms we compute the location-aware frequency score or spatio-temporal frequency score between the term and the query. This approach, however, will be very expensive for a large number of geo-textual messages. For efficiently processing a query, we need to address the following challenges. First, it is computationally expensive to return the exact result of L*k*TQ or ST*k*TQ. Hence, we need seek approximate solutions with high accuracy. Second, we need to measure term frequency, term location proximity, and term freshness in a continuous fashion. Therefore, it is non-trivial to propose a hybrid indexing structure and its corresponding algorithm that could effectively prune the search space based on term frequency, location proximity , and freshness simultaneously. Because of the sliding-window scenario of L*k*TQ and ST*k*TQ, the indexing mechanism must be able to handle geo-textual data streams with high arrival rate.

This paper expands on a previous study [55]. In particular, based on the L*k*TQ we define a novel query ST*k*TQ for discovering bursty and trending terms over a stream of geo-textual objects. The ST*k*TQ additionally takes temporal aspect into account. We introduce a matric for measuring the spatio-temporal popularity of a term within a sliding window, which considers term frequency, spatial proximity, and term freshness. Unlike the metric of L*k*TQ , the scoring function of ST*k*TQ (spatio-temporal popularity score) is changing over time as time elapses. To efficiently compute spatio-temporal popularity score, we develop a backdate mapping method that capable of preventing the score from being re-computed over time. Next, we propose a best-first search algorithm to efficiently process the ST*k*TQ. We also report on experiments that offer insight into the efficiency performance of our proposed algorithm for processing ST*k*TQ in different settings.

Our contributions are summarized as follows:

1. We define a new problem of processing L*k*TQ that searches for the top-*k* locally popular terms by taking into account both term frequencies and location proximities from geo-textual dataset.
2. For taking term freshness into account, we propose a new query named Spatio-Temporal Top-*k* Term Query (ST*k*TQ) that searches for the top-*k* locally trending terms by considering term frequencies, location proximities, and term freshness from geo-textual dataset.
3. A hybrid quad-tree based indexing structure that has low storage and update cost and a searching algorithm with effective pruning strategies are proposed to enable the fast and accurate top-*k* term search. Specifically, since it is impossible to store every messages in such a big streaming data, we augment each quad-tree node with a summary file for summarizing the term frequencies. The non-leaf node maintains an upper bound error by storing the merging summaries of its child nodes. Misra-Gries summary (MG summary) [27] and Space-Saving summary (SS summary) [25, 26] are two simple and popular summaries for frequency estimation and heavy hitters problems. Due to the merge processing [1] of MG summaries is lightweight and has a guarantee on the accuracy of frequency [47], and there are a lot of merging manipulations in quad-tree nodes, we adopt the MG summary instead of the SS summary.

The rest of this paper is organized as follows: In Section 2, preliminaries and some related works are introduced. In Section 3, we provide our proposed solution on the problem. An experimental analysis is presented in Section 4. Section 5 provides a discussion and a conclusion is presented in Section 6.

2 Preliminaries and related work

2.1 Top-*k* spatial querying

Top-*k* spatial-keyword query (e.g., [3–5, 10, 15, 29, 59–61, 66]) retrieves *k* most relevant geo-textual objects by considering both location proximity (to query location) and textual similarity (to query keywords). Hybrid indices are developed to store the location and text information of objects, which use both location information and text information to prune search space during the query time. Most of such indices combine spatial index (e.g., R-tree, quad-tree) and the inverted file for storing location and text information, respectively. However, these studies aim at retrieving top-*k* geo-textual objects, which is different from the problem of retrieving top-*k* terms. Shang et. al. extended this idea to trajectory data and studied spatial-keyword trajectory search [34, 63].

2.2 Frequent item counting

In stream data processing, aggregation is a widely studied problem. Existing aggregation techniques are commonly categorized into counter-based techniques and sketch-based techniques.

Counter-based techniques monitor all the items with a fixed number of counters, each message for an individual counter in a subset of S. When an item in the monitored set comes, its counter is updated. If the item is not in the monitored set and the counters are full, then some other actions will be taken in different algorithms. For instance, Space-Saving algorithm can find any item with the minimum counter value, replace the new item with it, and then increase the counter by 1.

Another popular algorithm - MG summary is very simple to implement. Given a parameter k, since an MG summary stores $k - 1$ (item, count) pairs, there are three cases when dealing with a new coming item i in the stream.

1. if i has already maintained in the current counters, increase its counter value by 1;
2. if i is not in the monitoring list and the number of counters does not reach k, insert i into the summary and set its counter value to 1;
3. if i is not in the monitoring list and the summary has maintained k counters, we decrement all the counter value of messages in the monitored set by 1 and remove all the messages whose counter value is equal to 0.

Other notable counter-based algorithms include LossyCounting [24] and Frequent [8, 14].

Sketch-based techniques monitor all the messages rather than a subset of S using hashing techniques. Messages are hashed into the space of counters, and the hashed-to counters will be updated for every hit of the corresponding item. The CountSketch algorithm [51] solves the problem of finding approximate top keywords with success probability (1-δ). The GroupTest algorithm [7] aims at searching queries about hot items and achieves a constant probability of failure, δ. And it is generally accurate. Count-Min Sketch [6] is also a representative Sketch-based technique.

Sketch-based techniques have less accuracy and less guarantees on frequency estimation than counter-based techniques due to hashing collision. Moreover, they do not provide guarantee about relative order in the continuous stream. Therefore, we adopt counter-based techniques in our work.

2.3 Systems for term-based searching

There are several recent systems using related techniques. Skovsgaard [47] designs a framework supporting indexing, updating and query processing which are capable of return the top-k terms in posts in a user-specified spatio-temporal range. The called adaptive frequent item aggregator (AFIA) system is implemented through multiple layers of grids to partition space into multiple granularities. In each grid cell, a precomputed summary is maintained. The system also performs a checkpoint to prevent the situation where a counter enters the top-k counters along with its possible error as a standalone system employing spatial-temporal indexing.

BlogScope [2] is a system which collects news, mailing list, blogs, and so on. It supports finding and tracking the objects, events or stories in real world, monitoring most of the hot keywords as well as the temporal/spatial bursts. The biggest drawback of BlogScope is that it cannot aggregate keywords according to user-specified spatio-temporal region. Moreover, it has weak timeliness which only support the search in a few minutes.

NewsStand [48] and TwitterStand [30] are two similar systems. NewsStand is a news aggregator of spatio-textual data, collecting geographical contents from RSS feeds into story clusters. Users are expected to retrieve and search some stories related to the query keywords within the geographical region. The difference between NewsStand and TwitterStand is that TwitterStand uses Tweets as data source instead of RSS feeds. They both adopt a spatio-textual search engine, which supports spatio-temporal searching not long, on a small ProMED dataset. However, both of the systems have a not high rate of updating.

In the next step, it is of interest to integrate spatial, temporal, and textual data to define novel queries. First, we may integrate trajectory data [17–22, 37, 44–46, 64, 67] with textual data and to conduct novel spatio-textual trajectory search, recommendation, and analysis studies. Second, we may use POI (points of interest) data and geo-tagged social media data to discover hot regions and locations [11, 31–33, 39, 40, 43, 49, 57]. Third, we may also study how to integrate textual data to routing problems [35, 36, 38, 41, 42, 52, 56, 68] as well as spatio-textual routing problem. Fourth, we may study how to integrate streaming data sampling methods and pattern analysis [12, 13, 16, 23, 50, 53, 54, 58, 65] integrating with spatio-textual data.

3 Problem statement

3.1 Geo-textual message

Let D be a 2D Euclidean space, W be a sliding window, S be a set of geo-textual messages located within D and W. Each geo-textual message is denoted by $o = (pos, text, time)$, where pos is a point location in D, $text$ is text information, and $time$ denotes timestamp.

3.2 Location-aware top-*k* term query (L*k*TQ)

An L*k*TQ q is represented by a tuple (loc, k) where loc indicates the query location and k denotes the number of result terms. It returns k terms with the highest *location-aware frequency score* of messages within W.

The location-aware frequency score of a term t in the sliding window W is defined as a linear combination of the distance and the frequency of the term in W:

$$FS(t) = \alpha \times \frac{freq(t)}{|W|} + (1-\alpha) \times (1 - \frac{d(q, W_t)}{d_{diag} \times |W_t|}) \tag{1}$$

where $freq(t)$ is the number of messages containing term t, $|W|$ is the number of messages in the sliding window W, $d(q, W_t)$ is the sum of distance between the query and the messages that contain t in window W, d_{diag} is the diagonal length of the rectangular region R, $|W_t|$ denotes the number of messages in W that contain t, and α $(0 \le \alpha \le 1)$ is a parameter which balances the weight between the term frequency and the location proximity.

3.3 Spatio-temporal top-*k* term query (ST*k*TQ)

An ST*k*TQ q is represented by a tuple (loc, k) where loc indicates the query location and k denotes the number of result terms. It returns k terms with the highest *spatio-temporal popularity score* of messages within W.

The spatio-temporal popularity score of a term t at timestamp δ in the sliding window W is defined as follows:

$$PS(t, \delta) = (\alpha \times \frac{freq(t)}{|W|} + (1-\alpha) \times (1 - \frac{d(q, W_t)}{d_{diag} \times |W_t|})) \times TS(\delta) \tag{2}$$

where $freq(t)$ is the number of messages containing term t, $|W|$ is the number of messages in the sliding window W, $d(q, W_t)$ is the sum of distance between the query and the messages that contain t in window W, d_{diag} is the diagonal length of the rectangular region

R, $|W_t|$ denotes the number of messages in W that contain t, α ($0 \leq \alpha \leq 1$) is a parameter which balances the weight between the term frequency and the location proximity, and $TS(\delta)$ denotes the temporal score of term t at timestamp δ, which is computed as (3):

$$TS(\delta) = \frac{1}{|W_t|} \times \sum_{o \in W_t} \times D^{-(\delta - o.time)}, \tag{3}$$

where D is base value that reflects the rate of the freshness decay. Based on (3) we can find that the value of $TS(\delta)$ is monotonically decrease with $t_e - o.t_c$. It is widely applied as a popular measurement of recency for streaming items (i.e., we prefer the fresh items in the sliding window). According to the experimental evaluations [9], the exponential decaying function is capable of effectiveness in blending the freshness and text similarity of messages.

Theorem 1 *Equation (2) guarantees that the relative ranking of two different objects w.r.t. a query is consistent as time elapses. Specifically, if $PS(t, \delta) > PS(t', \delta)$, then $\forall \Delta > 0$ we have $PS(t + \Delta, \delta) > PS(t' + \Delta, \delta)$.*

Proof Based on (3) and the assumption that $PS(t, \delta) > PS(t', \delta)$, we have $(\alpha \times \frac{freq(t)}{|W|} + (1 - \alpha) \times (1 - \frac{d(q, W_t)}{d_{diag} \times |W_t|})) > (\alpha \times \frac{freq(t')}{|W|} + (1 - \alpha) \times (1 - \frac{d(q, W_{t'})}{d_{diag} \times |W_{t'}|}))$. Because $\Delta > 0$, we have $D^{-(\delta - o.time)} > D^{-(\delta + \Delta - o.time)}$. So we complete the proof. □

4 Proposed solution

The details of our algorithm is presented in this section for handling top-*k* query. Specifically, we first introduce the data indexing model in the algorithm to store all the data items (Section 3.1). Then, we show the process of the query searching in Section 3.2, including the best-first algorithm which explain how we can get the terms with highest scores.

4.1 Data indexing model

We use a data structure of quad-tree to store all the spatial-text data items in stream for faster indexing. Quad-tree is a tree-like data structure, whose basic idea is to divide the

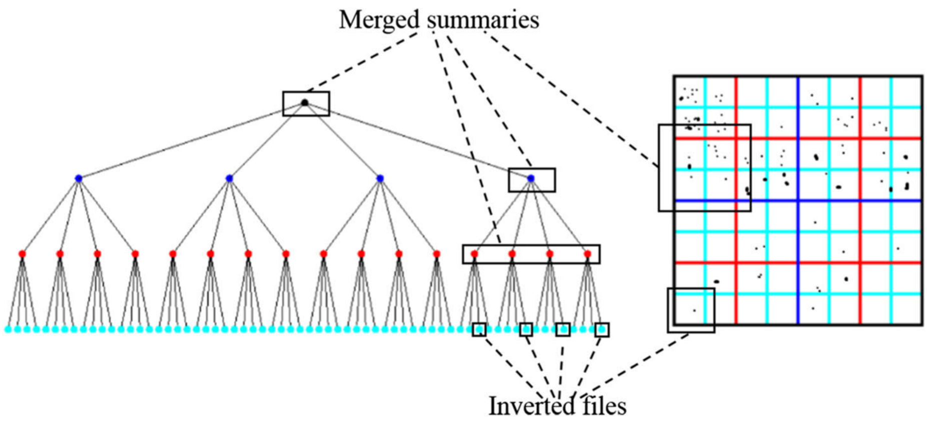


Figure 3 The basic structure of a quad-tree

geography space into different levels of tree structures. It divides the space of known range into four equal subspaces, recursively until the tree reaches a certain depth or stops after a certain requirement. Quad-tree is widely used in image processing, spatial data indexing, fast collision detection in 2D, sparse data, and so on. The basic structure in our algorithm is shown in Figure 3. One thing to mention, the different color of the nodes corresponds to the certain quadrant areas in the rectangle of right side.

Quad-tree has very simple structure, and when the spatial data object distribution is relatively uniform, it has relatively high insertion and query efficiency. The black points in the figure is the data items which locate in their expected region. In our algorithm, we set M as the largest number of data items in which a leaf node will contain. That means, if the number of items stored in a node is more than M, this node will be split into four nodes with an equal size. This is our stop splitting condition of a node instead of fixing tree depth.

4.2 Processing of L*k*TQ

According to our problem definition in Section 2.1, we proceed to describe the framework we use to get top-k terms with the highest scores rapidly and accurately in a specified situation adapting the continuous increases of the social geo-tagged stream data.

4.2.1 Overview

Different from the conventional spatial querying algorithm, the location component of L*k*TQ is a point instead of a specified spatial region. We aim to find the most relevant k terms in a comprehensive consideration with distance and frequency. Since we maintain a fixed number of items over a sliding window, so when an item in the stream comes and is inserted, an item with the oldest time stamp should be deleted.

Algorithm 1 Misra-Gries(counters k, stream T)

```
$n \leftarrow 0$;
$T \leftarrow \emptyset$;
foreach $i$ do
    $n \leftarrow n + 1$;
    if $i \in T$ then
        $c_i \leftarrow c_i + 1$;
    else if $|T| < k - 1$ then
        $|T| \leftarrow |T| \cup \{i\}$;
        $c_i \leftarrow 1$;
    else
        for all $j \in T$ do
            $c_j \leftarrow c_j - 1$;
            if $c_j = 0$ then
                $|T| \leftarrow |T| \backslash \{j\}$
```

4.2.2 Summary merging

Each quad-tree leaf node stores the summaries of all the textual information of contained items. Agarwal et al. [1] has proved that MG summary and SS summary are isomorphic and SS summary can be transferred by MG summary. Recall that, for the reason that the merge processing of MG summaries is easy and efficient, while there are a lot merging manipulations in quad-tree, we adopt the MG summary instead of the SS summary. The process of merging MG summaries is pretty simple. First combine two summaries by adding up the corresponding counters. This step will result in up to $2k$ counters. Then a prune operation is manipulated: take the $(k + 1)$-th largest counter, and subtract its counter value from all the counters, and finally remove all the non-positive counters. This is a process with constant number of sorts and scans of summaries of size $O(k)$. The details of the algorithm is shown in Algorithm 1.

In this algorithm, all the nodes including leaf nodes and parent nodes store summaries of the items in it. In leaf nodes, summaries are computed using the process stated in Algorithm 1, while in parent nodes, summaries come from the merging processing of all its child node using the method we describe above.

4.2.3 Computing the location-aware frequency score

Given a term, to obtain its score, we have two steps:

1. first we need to compute the score in each node employing the summaries stored in each node. As we define in Section 2.1, the formula to calculate the score is:

$$score(t) = \alpha * \frac{fre}{|S|} + (1 - \alpha) * (1 - \frac{d}{d_{diag}})$$

   For convenience, we divide and score calculation formula as the "Frequency part" ($\frac{fre}{|S|}$) and the "Distance part" ($1 - \frac{d}{d_{diag}}$). Essentially, the score is a linear integration of the two parts. As the MG summaries estimate the frequency of any item with error at most $n/(k + 1)$ (n is the number of all the items), we add the maximum error to fre to calculate the "Frequency part". d is defined as the distance between the query and the object which contains the term, here, we use the minimum distance between the query and the four edges of the node which contains the term as an upper bound value.

   Since a term may occur more than one time in a node, we need to consider the redundant calculation of the same term in distance. For each part, we recursively compute the maximum part score from the quad-tree root, after that, the "Distance part" needs to be divided by the number of the the same term shown in the same node. Finally, we calculate the sum of the two parts with a linear balance parameter α. Obviously, in this way, we get an upper bound score for each term in each node.
2. After we get all the scores in each node for a term, the score of the term can be integrated. It is computed by adding the score of several nodes to make the score value as big as possible. The rule must be kept that the nodes involved should cover the whole area of the given region (the quad-tree).

4.2.4 Best first querying algorithm

Pseudo code of the detailed algorithm implementation is provided in Algorithm 2.

α is a weight parameter to balance the distance and frequency in computing score. This parameter will be varied in our experiments to validate the influence. C in Line 2 is a priority queue storing all the candidate words. To get the candidate terms, we extract the summaries in root node of the quad-tree. However, we consider the situation, if the candidates are thousands level while the user specified k is only a very small number, then the redundant computing of thousands term scores will cost a lot and waste too much time. So we come up with a pruning method to avoid this while ensuring that we will not miss any real valuable candidate terms.

The pruning process is as follows: after we get the exact k from user, we recompute the score of the k-th term, making the score of "distance part" to 0 as a lower bound. Then, from the $(k + 1)$-th term in the root summary (since the summary are all sorted), we recompute their score of "distance part" as full value as their upper bound. When i-th $(i > k)$ term has an upper bound score which is smaller than the lower bound score of the $k-$th term, we believe that all the words after the i-th term have no possibilities to get onto the top of the priority queue in the near future of k times of manipulation of Line 4-13 in Algorithm 2.

Algorithm 2 GetTopKTerms(QuadTree tree, Query query)

```
$\alpha \leftarrow ALPHA$;
$C \leftarrow a\ priority\ queue\ storing\ candidate\ words$;
$Result \leftarrow the\ list\ storing\ the\ top - k\ results$;
foreach $i = C.poll()\&\&Result.size() < query.getK()$ do
    $tree.traverse(root, node)$ :
    $i.score' \leftarrow score\ of\ i\ in\ one\ of\ node's\ children$;
    $i.score' \leftarrow getWordScore(tree, query, i, \alpha)$;
    if $i.score' < i.score$ then
        $replace(i.score, i.score')$;
    $until\ node\ is\ Leaf\ and\ get\ the\ exact\ score : i.score\_exact$;
    $C.add(i)$;
    if $C.poll().score == i.score\_exact$ then
        $Result.add(i)$;
Return $Result$;
```

Line 4-13 shows the process to find the exact score of a term. For each candidate which is popped from the top of the priority queue, we traverse the whole tree from the root to leaf nodes. If we find a smaller score in a child node than in the parent node, we replace the current score with the new smaller score and insert the new score into the queue until we get a small enough score which is equal to the top element in the priority queue. Then, this term with an exact score will be added in our result list - see Line 12-13.

4.3 Processing of ST*k*TQ

We proceed to present our method for processing ST*k*TQ. The framework of processing ST*k*TQ is similar to the framework of processing L*k*TQ . However, we need to compute spatio-temporal popularity score instead of location-aware frequency score. Differently from location-aware frequency score, the spatio-temporal popularity score of a term is changing over time. It is computational prohibitive to maintain an up-to-date spatio-temporal popularity score by re-computation. To address this challenge, we propose a novel method to compute spatio-temporal popularity score without re-computation, which is presented as follow.

Computing the spatio-temporal popularity score Based on (2), to avoid re-computation of spatio-temporal popularity score of a given term we need to make $TS(\delta)$ consistent. So we develop a backdating technique that maps the spatio-temporal popularity score to a constant timestamp. In particular, we first split the spatio-temporal popularity score into a set of elements. Each element is the score value that is contributed by an object.

For each score element, we backdate its scoring value to the earliest timestamp of the sliding window. However, we have to process a large number of score elements, and each score element may have different object timestamps.

Definition 1 Score Element: Given a term t, a timestamp δ, and a geo-textual object o that contains t, the score element contributed by o for term t at timestamp δ is calculated as follows:

$$Element(t, \delta, o) = FS(t) \times \times \frac{1}{|W_t|} \times D^{-(\delta - o.time)}. \tag{4}$$

Different objects may have different timestamp, which make it difficult to compute the score element because we have to compute them one by one based on unique timestamp for each object. So we convert $Element(t, \delta, o)$ of different objects into a score element that has a unique timestamp, which is called backdate timestamp. Definition 2, instead of the subscription update threshold, where t_b is a subscription-independent backdated time.

Definition 2 Backdate Timestamp Score Element. Let o be an object, the backdate tiemstamp score element at time δ is defined as (5)

$$Element(t, \delta, o) = FS(t) \times \times \frac{1}{|W_t|} \times D^{-(\delta - b)}, \tag{5}$$

where b is a backdate timestamp, which is the earliest time of the current sliding window.

5 Experimental study

We conduct experiments to evaluate the solution and to compare with other feasible methods. All the experiments are conducted on a workstation with Intel(R) Xeon(R) CPU $E5 - 2643$ 0 @3.30GHz and 64 GB main memory on a 64-bit Windows operating system. And the whole framework is implemented in Java.

The dataset consists of tweets collected in the United States. It has 20,000,000 messages, and each of them contains a timestamp, a list of terms, and the longitude and latitude of the tweet (i.e., the geographical tag set by user). Notice that the result of each set of experiments is averaged over 10 independent trails with different query inputs.

5.1 Baselines

We use the following exact algorithm as the baseline for making comparison and validation of our approach. The indexing structure of the baseline is also based on a quad-tree. Specifically, in each leaf node of the quad-tree, we store the exact frequency of each term. When a new message arrives, we update the frequency in the corresponding node. To get the frequency information of a non-leaf node, we traverse the quad-tree recursively until we reach the leaf node. This approach can return the exact result of both LkTQ and STkTQ. Therefore, it can be used as a measure of querying accuracy in subsequent experiments.

5.2 Index updating of quad-tree

First, we conduct an experiment to evaluate the performance when we insert and remove a message in the sliding window. Because we aim to find the top-k term over a sliding window, if the sliding window is full, when a new message is generated, an old message will be

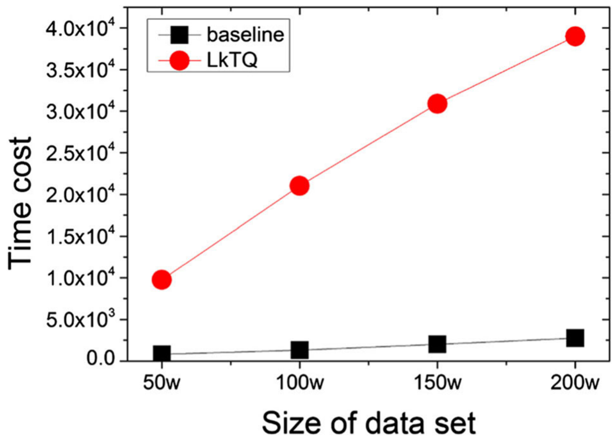


Figure 4 Time cost of updating index on varied size of data for L*k*TQ

deleted. Experimental results are shown in Figures 4 and 5. We can see that the two manipulations in baseline and our approach scarcely cost time as the index updating process are included. This is based on a well-constructed quad-tree. Therefore, we conduct an experiment to find out the time cost of constructing a quad-tree with all the term frequencies computed and index updating.

Specifically, for the baseline method, constructing the quad-tree includes counting and merging all the term frequencies. On the other hand, for our approach, the construction stage includes computing all the MG summaries of all the nodes in the quad-tree. We see that the time cost of index construction of our approach is higher than that of the baseline method. In the following experiments, we show that our approach can achieve a much higher query efficiency.

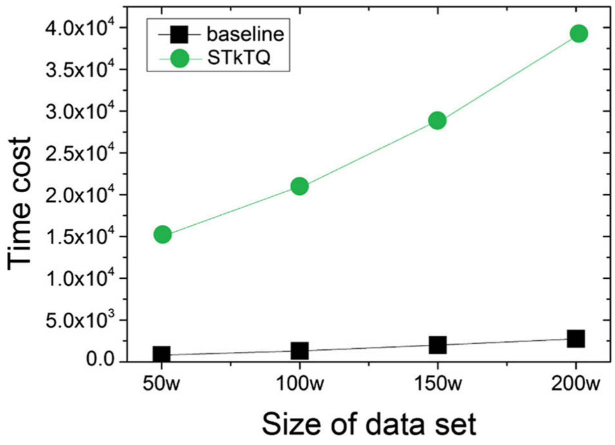


Figure 5 Time cost of updating index on varied size of data for ST*k*TQ

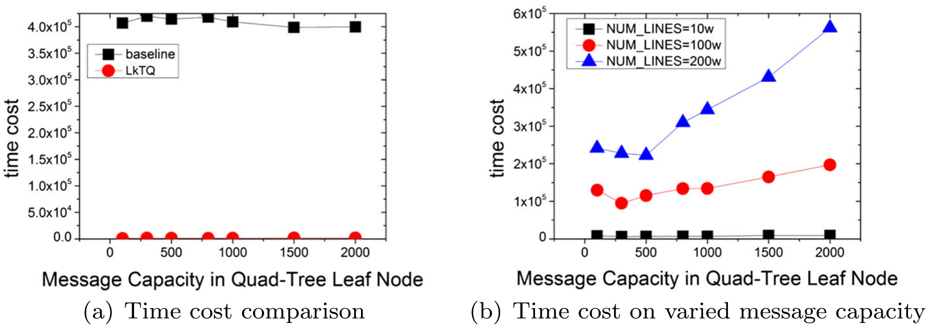


Figure 6 Varying Message Capacity in Quad-Tree Leaf Node

5.3 Varying message capacity in quad-tree leaf node

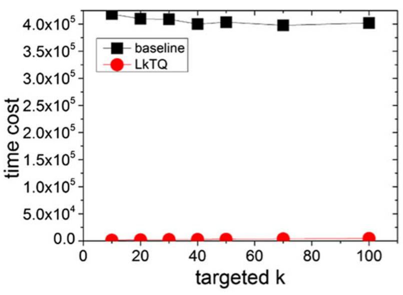
Recall that when we construct a quad-tree to index all the messages, we have a condition to determine when we split the node and generate new child nodes. The condition is that when the number of messages in a node reaches M, the node should be split. We conduct an experiment to vary the maximum number of messages stored in a leaf node, so that we can find out what is the best message capacity of leaf node with better performance. Other parameter settings are: the targeted k is 20, $\alpha = 0.7$, and the number of counters in MG summary is 500. Specially, the number of counters is set to 500 mainly for large data sets to reduce summary error.

Figure 6 shows the experimental results. Figure 6a shows the comparison results when the data set amount is 10,000. M is ranged from 100 to 2000. Our approach is much faster than the baseline method. It has a little fluctuation in varying M. The message capacity of quad-tree leaf node has no big influence on the performance in baseline. Once M is fixed, the tree is fixed and the score is stable to compute. However, M actually influences the performance of our algorithm. In theory, the bigger the M is, the smaller the depth of the quad-tree is. Because when computing score in each node, we use the nearest edge to the query in “Distance part”. If the tree is deeper, the distance will be smaller and the number of leaf node is larger. As shown in Figure 6b, when M increases, time cost is higher because when M is getting larger, the cost of splitting is larger. There is a little turning down when M is around 300 and 500. And in this range, it has almost the best performance.

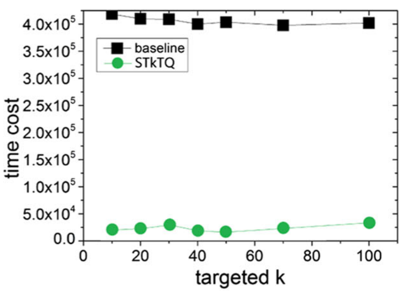
5.4 Varying targeted *k*

In this experiment, we vary the targeted k (see Figure 7). The targeted k is actually specified by users, and other fixed parameters are set as follows: $\alpha = 0.7$, the maximum number of messages in each leaf node M is 1,000, and the number of counters in MG summary is 100. Although M around 300 to 500 has the most excellent results, 1,000 is chose for controlling the quad-tree depth and for more accurate results. Because, experiments are conducted to prove that, when M is close to 1,000, the results will be consistent when other parameters varied.

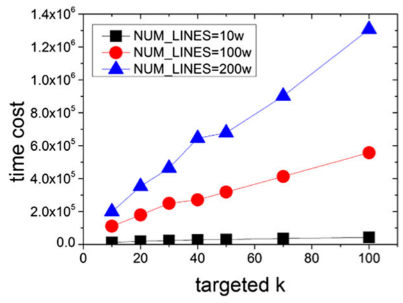
Figure 7 shows the results. The range of targeted k is set according to the normal requirements of users. The performance of our algorithm is remarkably better than the baseline, which counts one by one (Figure 7a and b). The amount of data set in Figure 7a and b is 10,000, however, baseline need approximately seven minutes to return the results. The



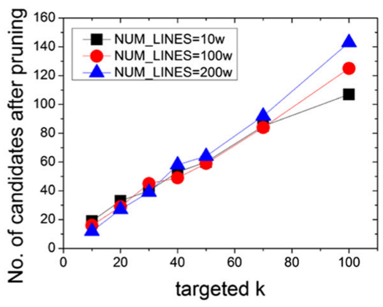
(a) Time cost comparison for LkTQ



(b) Time cost comparison for STkTQ



(c) Time cost on varied amount of data



(d) No. candidates after pruning

Figure 7 Varying targeted *k*

time cost of baseline is on a stable and inefficient level which is around 400,000 ms. For larger data sets, baseline has an extremely slow running speed. For instance, dealing with 5,000 messages, it needs about 12 million milliseconds and it costs nearly 60 million milliseconds to handle 100,000 messages, which is very inefficient. So we do not show the non-competitive results.

Actually, as expected, the time cost of our approach increases as the target-*k* increases. It is not obvious to see for the great disparity of the time cost on tick labels. Therefore, another further experiment has proved this shown in Figure 7b. Moreover, as the amount of data set is getting larger, the tendency is more conspicuous. Specifically, to find out the origin of the fastness, we conduct another experiment to validate the number of candidates after our pruning algorithm according to *k* is truly close to *k*. The result is shown in Figure 7c as proof.

5.5 Accuracy versus baseline

Accuracy is a vital factor which users concern. The accuracy experiment results of our algorithm versus baseline are shown in Figure 8. We measure the fraction of the correct top-*k* returning from our algorithm for different amount of data sets. Since the baseline has such inefficient running speed, we choose relatively small data sets, which however, does not influence the high performance of our algorithm. When targeted-*k* is set to a low value, our approach produces pretty accurate results and can guarantee 80% correctness. As targeted-*k* becomes large, the accuracy is a little decreasing. However, the lowest accuracy is above 0.39 even when targeted-*k* is 100 and enable satisfy most of users' requirements.

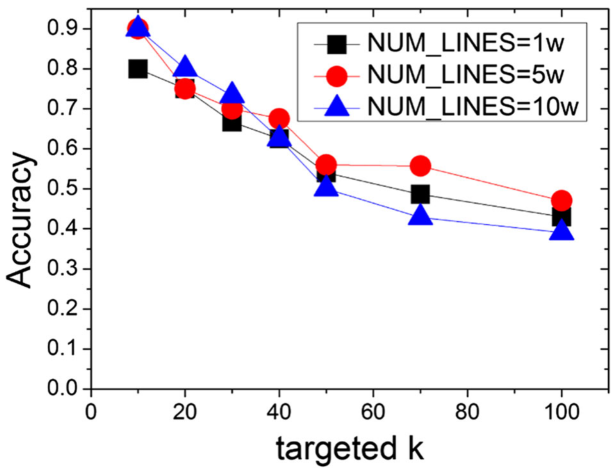


Figure 8 Accuracy on varied amount of data

5.6 Varying parameter α

α is a parameter which balances the weight of the score computing formula. Varying parameter α is to adjust the influence rate of distance and term frequency. It depends on users to determine their preferences. Through experiments, it is proved that the results of our algorithm are sensitive in a range of (0.9, 1.0). Certainly, when α is set to 0 or 1, then the results represent the unilateral influence of distance or frequency. Specifically, the sensitive range of α is influenced by the distribution of data sets. However, the experiments we conduct prove that our algorithm can be sensitive to the results by varing α so that it can satisfy the preferences of users.

6 Conclusions

We propose a new approach for supporting querying the local top-k most frequent, popular, and trending terms in social stream data with a huge amount of geo-tagged tweets. A comprehensive definition of term score considering both distance with queries and the term frequencies is presented. Quad-tree is used for indexing and extended to employ MG summaries to count term frequencies rapidly. Query processing adopts a best-first algorithm to pick up candidate terms and obtain exact term score for results. An empirical experiment is conducted to validate our algorithm and offers performance and accuracy of top-k term querying in spatial-textual social data streams and the framework is capable of returning results accurately and rapidly.

References

1. Agarwal, P.K., Cormode, G., Huang, Z., Phillips, J.M., Wei, Z., Yi, K.: Mergeable summaries. ACM Trans. Database Syst. **38**(4), 26,1–26,28 (2013)

2. Bansal, N., Koudas, N.: Blogscope: a system for online analysis of high volume text streams. In: VLDB, pp. 1410–1413 (2007)
3. Chen, L., Shang, S.: Approximate spatio-temporal top-k publish/subscribe. WWW J., online first: 1–23 (2018)
4. Chen, L., Shang, S., Zhang, Z., Cao, X., Jensen, C.S., Kalnis, P.: Location-aware top-k term publish/subscribe. In: ICDE, pp. 1–12 (2018)
5. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial Web objects. PVLDB **2**(1), 337–348 (2009)
6. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithms **55**(1), 58–75 (2005)
7. Cormode, G., Muthukrishnan, S.: What's hot and what's not: tracking most frequent items dynamically. ACM Trans. Database Syst. **30**(1), 249–278 (2005)
8. Demaine, E.D., Lòpez-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: ESA, pp. 348–360 (2002)
9. Efron, M., Golovchinsky, G.: Estimation methods for ranking recent information. In: SIGIR, pp. 495–504. ACM (2011)
10. Felipe, I.D., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: ICDE, pp. 656–665 (2008)
11. Guo, D., Zhu, Y., Xu, W., Shang, S., halls, Z.Ding.: How to find appropriate automobile exhibition Towards a personalized recommendation service for auto show. Neurocomputing **213**, 95–101 (2016)
12. Han, J., Zheng, K., Sun, A., Shang, S., Wen, J.: Discovering neighborhood pattern queries by sample answers in knowledge base. In: ICDE, pp. 1014–1025 (2016)
13. Hu, S., Wen, J., Dou, Z., Shang, S.: Following the dynamic block on the Web. World Wide Web **19**(6), 1077–1101 (2016)
14. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. ACM Trans. Database Syst. **28**, 51–55 (2003)
15. Li, Z., Lee, K.C.K., Zheng, B., Lee, W., Lee, D.L., Ir-tree, X.Wang.: An efficient index for geographic document search. IEEE Trans. Knowl. Data Eng. **23**(4), 585–599 (2011)
16. Li, Z., Shang, S., Xie, Q., Zhang, X.: Cost reduction for Web-based data imputation. In: DASFAA, pp. 438–452 (2014)
17. Liu, K., Yang, B., Shang, S., Li, Y., Ding, Z.: MOIR/UOTS: trip recommendation with user oriented trajectory search. In: MDM, pp. 335–337 (2013)
18. Liu, K., Li, Y., Dai, J., Shang, S., Zheng, K.: Compressing large scale urban trajectory data. In: CloudDP@EuroSys, pp. 3:1–3:6 (2014)
19. Liu, K., Li, Y., Ding, Z., Shang, S., Zheng, K.: Benchmarking big data for trip recommendation. In: ICCCN, pp. 1–6 (2014)
20. Liu, J., Zhao, K., Sommer, P., Shang, S., Kusy, B., Jurdak, R.: Bounded quadrant system: error-bounded trajectory compression on the go. In: ICDE, pp. 987–998 (2015)
21. Liu, J., Zhao, K., Sommer, P., Shang, S., Kusy, B., Lee, J., Jurdak, R.: A novel framework for online amnesic trajectory compression in resource-constrained environments. IEEE Trans. Knowl. Data Eng. **28**(11), 2827–2841 (2016)
22. Liu, A., Wang, W., Shang, S., Li, Q., Zhang, X.: Efficient task assignment in spatial crowdsourcing with worker and task privacy protection. GeoInformatica, online first: 1–28 (2017)
23. Liu, A., Shen, X., Li, Z., Liu, G., Xu, J., Zhao, L., Zheng, K., Shang, S.: Differential private collaborative Web services qos prediction. WWW J., online first: 1–24 (2018)
24. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. PVLDB **5**(12), 1699 (2012)
25. Metwally, A., Agrawal, D., El Abbadi, A.: Efficient computation of frequent and top-k elements in data streams. In: ICDT, pp. 398–412 (2005)
26. Metwally, A., Agrawal, D., El Abbadi, A.: An integrated efficient solution for computing frequent and top-k elements in data streams. ACM Trans. Database Syst. **31**(3), 1095–1133 (2006)
27. Misra, J., Gries, D.: Finding repeated elements. Sci. Comput. Program. **2**(2), 143–152 (1982)
28. Ozsoy, M.G., Onal, K.D., Altingovde, I.S.: Result diversification for tweet search. In: WISE, pp. 78–89 (2014)
29. Rocha-Junior, J.B., Gkorgkas, O., Jonassen, S., Nørvåg, K.: Efficient processing of top-k spatial keyword queries. In: SSTD, pp. 205–222 (2011)
30. Sankaranarayanan, J., Samet, H., Teitler, B.E., Lieberman, M.D., Sperling, J.: Twitterstand: news in tweets. In: SIGSPATIAL, pp. 42–51 (2009)
31. Shang, S., Deng, K., Xie, K.: Best point detour query in road networks. In: ACM SIGSPATIAL, pp. 71–80 (2010)
32. Shang, S., Yuan, B., Deng, K., Xie, K., Zhou, X.: Finding the most accessible locations: reverse path nearest neighbor query in road networks. In: ACM SIGSPATIAL, pp. 181–190 (2011)

33. Shang, S., Yuan, B., Deng, K., Xie, K., Zheng, K., Zhou, X.: PNN query processing on compressed trajectories. GeoInformatica **16**(3), 467–496 (2012)
34. Shang, S., Ding, R., Yuan, B., Xie, K., Zheng, K., Kalnis, P.: User oriented trajectory search for trip recommendation. In: EDBT, pp. 156–167 (2012)
35. Shang, S., Lu, H., Pedersen, T.B., Xie, X.: Finding traffic-aware fastest paths in spatial networks. In: SSTD, pp. 128–145 (2013)
36. Shang, S., Lu, H., Pedersen, T.B., Xie, X.: Modeling of traffic-aware travel time in spatial networks. In: MDM, pp. 247–250 (2013)
37. Shang, S., Ding, R., Zheng, K., Jensen, C.S., Kalnis, P., Zhou, X.: Personalized trajectory matching in spatial networks, vol. 23 (2014)
38. Shang, S., Liu, J., Zheng, K., Lu, H., Pedersen, T.B., Wen, J.: Planning unobstructed paths in traffic-aware spatial networks. GeoInformatica **19**(4), 723–746 (2015)
39. Shang, S., Zheng, K., Jensen, C.S., Yang, B., Kalnis, P., Li, G., Wen, J.: Discovery of path nearby clusters in spatial networks. IEEE Trans. Knowl. Data Eng. **27**(6), 1505–1518 (2015)
40. Shang, S., Guo, D., Liu, J., Zheng, K., Wen, J.: Finding regions of interest using location based social media. Neurocomputing **173**, 118–123 (2016)
41. Shang, S., Chen, L., Wei, Z., Guo, D., Wen, J.: Dynamic shortest path monitoring in spatial networks. J. Comput. Sci. Technol. **31**(4), 637–648 (2016)
42. Shang, S., Chen, L., Wei, Z., Jensen, C.S., Wen, J., Kalnis, P.: Collective travel planning in spatial networks, vol. 28 (2016)
43. Shang, S., Zhu, S., Guo, D., Lu, M.: Discovery of probabilistic nearest neighbors in traffic-aware spatial networks. World Wide Web **20**(5), 1135–1151 (2017)
44. Shang, S., Chen, L., Wei, Z., Jensen, C.S., Zheng, K., Kalnis, P.: Trajectory similarity join in spatial networks. PVLDB **10**(11), 1178–1189 (2017)
45. Shang, S., Chen, L., Jensen, C.S., Wen, J., Kalnis, P.: Searching trajectories by regions of interest, vol. 29 (2017)
46. Shang, S., Chen, L., Wei, Z., Jensen, C.S., Zheng, K., Kalnis, P.: Parallel trajectory similarity joins in spatial networks. VLDB J., online first: 1–26 (2018)
47. Skovsgaard, A., Sidlauskas, D., Jensen, C.S.: Scalable top-k spatio-temporal term querying. In: ICDE, pp. 148–159 (2014)
48. Teitler, B.E., Lieberman, M.D., Panozzo, D., Sankaranarayanan, J., Samet, H., Sperling, J.: Newsstand: a new view on news. In: SIGSPATIAL, pp. 18 (2008)
49. Wang, Y., Li, J., Zhong, Y., Zhu, S., Guo, D., Shang, S.: Discovery of accessible locations using region-based geo-social data. WWW J., online first: 1–18 (2018)
50. Wei, Z., Liu, X., Li, F., Shang, S., Du, X., Wen, J.: Matrix sketching over sliding windows. In: SIGMOD, pp. 1465–1480 (2016)
51. Wu, S., Lin, H., Hu, L., Gao, Y., Lu, D.: Finding frequent items in time decayed data streams. In: APWeb, pp. 17–29 (2016)
52. Xie, K., Deng, K., Shang, S., Zhou, X., Zheng, K.: Finding alternative shortest paths in spatial networks. ACM Trans. Database Syst. **37**(4), 29,1–29,31 (2012)
53. Xie, Q., Shang, S., Yuan, B., Pang, C., Zhang, X.: Local correlation detection with linearity enhancement in streaming data. In: CIKM, pp. 309–318 (2013)
54. Xie, X., Lu, H., Chen, J., Shang, S.: Top-k neighborhood dominating query. In: DASFAA, pp. 131–145 (2013)
55. Xu, Y., Chen, L., Yao, B., Shang, S., Zhu, S., Zheng, K., Li, F.: Location-based top-k term querying over sliding window. In: WISE, pp. 299–314 (2017)
56. Yang, B., Guo, C., Jensen, C.S., Kaul, M., Shang, S.: Stochastic skyline route planning under time-varying uncertainty. In: ICDE, pp. 136–147 (2014)
57. Yao, B., Chen, Z., Gao, X., Shang, S., Ma, S., Guo, M.: Flexible aggregate nearest neighbor queries in road networks. In: ICDE, pp. 1–12 (2018)
58. Yao, B., Zheng, W., Wang, Z., Chen, Z., Shang, S., Zheng, K., Guo, M.: Distributed in-memory analytics for big temporal data. In: DASFAA, pp. 1–16 (2018)
59. Zhang, C., Zhang, Y., Zhang, W., Lin, X.: Inverted linear quadtree: Efficient top k spatial keyword search. In: ICDE, pp. 901–912 (2013)
60. Zhang, D., Tan, K., Tung, A.K.H.: Scalable top-k spatial keyword search. In: EDBT, pp. 359–370 (2013)
61. Zhang, D., Chan, C., Tan, K.: Processing spatial keyword query as a top-k aggregation query. In: SIGIR, pp. 355–364 (2014)
62. Zhao, K., Chen, L., Cong, G.: Topic exploration in spatio-temporal document collections. In: SIGMOD, pp. 985–998 (2016)
63. Zheng, K., Shang, S., Yuan, N.J., Yang, Y.: Towards efficient search for activity trajectories. In: ICDE, pp. 230–241 (2013)

64. Zheng, K., Zheng, Y., Yuan, N.J., Shang, S.: On discovery of gathering patterns from trajectories. In: ICDE, pp. 242–253 (2013)
65. Zheng, K., Zheng, Y., Yuan, N.J., Shang, S., Zhou, X.: Online discovery of gathering patterns over trajectories. IEEE Trans. Knowl. Data Eng. **26**(8), 1974–1988 (2014)
66. Zheng, K., Su, H., Zheng, B., Shang, S., Xu, J., Liu, J., Zhou, X.: Interactive top-k spatial keyword queries. In: ICDE, pp. 423–434 (2015)
67. Zheng, B., Wang, H., Zheng, K., Su, H., Liu, K., Shang, S.: Sharkdb: An in-memory column-oriented storage for trajectory analysis. World Wide Web **21**(2), 455–485 (2018)
68. Zhu, S., Wang, Y., Shang, S., Zhao, G., Wang, J.: Probabilistic routing using multimodal data. Neurocomputing **253**, 49–55 (2017)