Location-Based Top-k Term Querying over Sliding Window

Ying Xu[1], Lisi Chen[2], Bin Yao[1(✉)], Shuo Shang[3(✉)], Shunzhi Zhu[4], Kai Zheng[5], and Fang Li[1]

[1] Shanghai Jiao Tong University, Shanghai, China
yaobin@cs.sjtu.edu.cn
[2] Hong Kong Baptist University, Hong Kong, China
[3] King Abdullah University of Science and Technology, Thuwal, Saudi Arabia
jedi.shang@gmail.com
[4] Xiamen University of Technology, Xiamen, China
[5] Soochow University, Soochow, China

Abstract. In part due to the proliferation of GPS-equipped mobile devices, massive svolumes 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 locally popular 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 both the term frequency and the proximities between the messages containing the term and the query location. 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.

Keywords: Top-k · Term · Location

1 Introduction

With the proliferation of social media, cloud storage, and location-based services, many researches [11,26–28,30,35,37] in spatial fields are studied. And 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

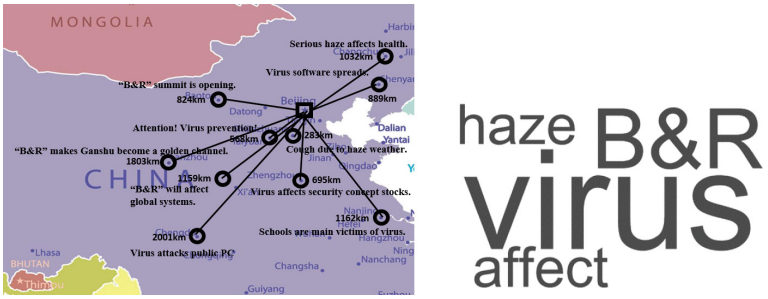
A. Bouguettaya et al. (Eds.): WISE 2017, Part I, LNCS 10569, pp. 299–314, 2017.
DOI: 10.1007/978-3-319-68783-4_21

in the stream if the user is not fast enough to discover it [17]; (2) *local-intended nature* - users from different locations may post messages related to diverging topics [46]. 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 [31] 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 a new kind of top-k term query, Location-based Top-k Term Query (LkTQ), that returns top-k locally popular terms by taking into account both location proximity and term frequency 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 are the k most locally popular terms based on a location-aware frequency score, which are shown in Fig. 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.



(a) Messages and distances (b) Tag Cloud

Fig. 1. Example of Querying in China

A straightforward approach for answering an LkTQ 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 between the term and the query. This approach, however, will be very expensive for a large number of geo-textual messages. For efficiently processing an LkTQ, we need to address the following challenges. First, it is computationally expensive to return the exact result of LkTQ. Hence, we need seek approximate solutions with high accuracy. Second,

the location-aware frequency score measures both term frequency and term location proximity 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 and location proximity simultaneously. Thrid, because of the sliding-window scenario of LkTQ, the indexing mechanism must be able to handle geo-textual data streams with high arrival rate.

Our contributions are summarized as follows:

1. We define a new problem of processing LkTQ that searches for the top-k locally popular terms by taking into account both term frequencies and location proximities from geo-textual dataset.
2. 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) [21] and Space-Saving summary (SS summary) [19,20] 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 [31], 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 Sect. 2, preliminaries and some related works are introduced. In Sect. 3, we provide our proposed solution on the problem. An experimental analysis is presented in Sect. 4. A conclusion is presented in Sect. 5.

2 Preliminaries and Related Work

2.1 Problem Definition

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 $o = (pos, text)$, where pos is a point location in D, and $text$ is text information. An LkTQ 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:

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

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 $(0 \leq \alpha \leq 1)$ is a parameter which balances the weight between the term frequency and the location proximity.

2.2 Related Work on Top-k Spatial-Keyword Query

Location-based queries [10,33,34,39,42], similarity search [11–13,38,41] or optimal path search [25,29,36] are hot issues in recent years. Top-k spatial-keyword query (e.g., [4,8,14,22,40,43–45]) is a more comprehensive problem of previous works. It 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.

2.3 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 [18] and Frequent [7,16].

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 [3] solves the problem of finding approximate top keywords with success probability $(1-\delta)$. The GroupTest algorithm [6] aims at searching queries about hot items and achieves a constant probability of failure, δ. And it is generally accurate. Count-Min Sketch [5] 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.

2.4 Related Systems

There are several recent systems using related techniques. Skovsgaard [31] 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 [32] and TwitterStand [24] 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.

3 Proposed Solution

The details of our algorithm are presented in this section for handling LkTQ. We first introduce the data indexing model, which is used to store all the data messages in a sliding windows (Sect. 3.1). Then, we show the process of the query searching in Sect. 3.2.

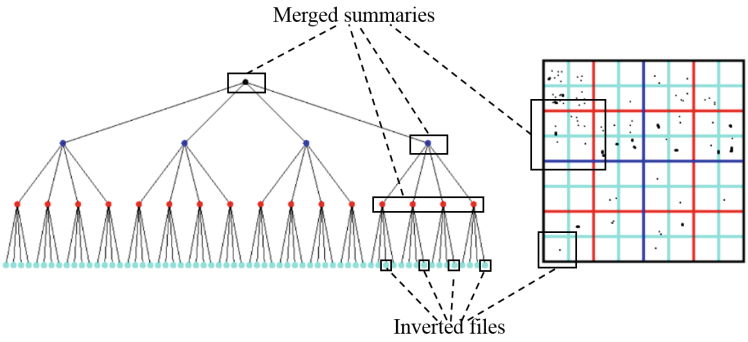


Fig. 2. The basic structure of a quad-tree.

3.1 Data Indexing Model

We use a quad-tree based indexing structure to store all the geo-textual messages in a stream for faster indexing. The basic idea of the quad-tree is to divide the underlying space into different levels of cells. Specifically, it recursively divides the space into four congruent subspaces, until the tree reaches a certain depth or stops on a certain condition. Quad-tree is widely used in image processing, spatial data indexing, fast collision detection in 2D, sparse data, etc. The basic structure in our algorithm is shown in Fig. 2. One thing to mention, the different color of the nodes corresponds to the certain quadrant cells in the rectangle of right side. And the information stored in leaf nodes are inverted files, while non-leaf nodes stores merged summaries.

Quad-tree has a very simple structure, and when the geo-textual messages distribution is relatively uniform, it has relatively high insertion and query efficiency. The black points in the figure is the messages which locate in their expected region. In our algorithm, we set M as the largest number of messages in a leaf node. In other words, if the number of messages stored in a node is more than M, this node will become a non-leaf node and be split into four leaf nodes with equal sizes.

3.2 Query Processing

Overview. According to our problem definition in Sect. 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 social geo-tagged stream data over a sliding window.

Index Updating over Sliding Window. Different from the region-based term query [31], the location of LkTQ is a point instead of a specified spatial region. We aim to find the locally most popular k terms in a comprehensive consideration with location proximity and frequency. If the sliding window is

not full, when a new message comes and is inserted to a leaf node of the quad-tree, the summaries in this node will be updated. Then, its parent node will update its merged summaries. The process will be done upward recursively until the root node of quad-tree gets the newest merged summaries. If the sliding window is full, when a message in the stream comes and is inserted, a message with the earliest time stamp should be deleted. Then, the index updating process is the same as the condition when the sliding window is not full.

Summary Merging. Each leaf node of quad-tree stores the summaries of all the textual information of contained messages. The details of MG algorithm [21] is shown in Algorithm 1. [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, we combine two summaries by adding up the corresponding counters. This step results 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. Finally, we remove all the non-positive counters. This is a process with constant number of sorts and scans of summaries of size $O(k)$.

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

```
1  n ← 0;
2  T ← ∅;
3  foreach i do
4      n ← n + 1;
5      if i ∈ T then
6          c_i ← c_i + 1;
7      else if |T| < k − 1 then
8          |T| ← |T| ∪ {i};
9          c_i ← 1;
10     else
11         for all j ∈ T do
12             c_j ← c_j − 1;
13             if c_j = 0 then
14                 |T| ← |T|\{j}
```

In this algorithm, both leaf nodes and non-leaf nodes store summaries of the messages 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.

Computing the Term Score. Given a term, we have two steps to obtain its score:

1. First, we need to compute the score in each node employing the summaries stored in each node. Eq. (1) defines the formula to calculate the score. For convenience, we divide the score calculation formula as the "Frequency part" ($\frac{freq}{|W|}$) and the "Distance part" ($1-\frac{d(q,W_t)}{d_{diag}\times|W_t|}$). 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 messages), we add the maximum error to $freq$ to calculate the "Frequency part". $d(q, W_t)$ is the sum of distance between the query and the messages that contains the term t, 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 of a message.
   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 calculation. Then, the "Distance part" involves the division part by the number of the messages that contain the same term in the same node. Finally, we calculate the sum of the two parts with a linear weight parameter α and normalize it into $[0, 1]$. 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).

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

Algorithm 2. GetTopKTerms(QuadTree tree, Query query)

```
1  C ← a priority queue storing candidate words;
2  Result ← the list storing the top k results;
3  foreach i = C.poll() AND Result.size() < query.getK() do
4      tree.traverse(root, node) :
5      i.score′ ← score of i in one of node′s children;
6      i.score′ ← getWordScore(tree, query, i, α);
7      if i.score′ < i.score then
8          replace(i.score, i.score′);
9      until node is leaf and get the exact score : i.score_exact;
10     C.add(i);
11     if C.poll().score == i.score_exact then
12         Result.add(i);
13 Return Result;
```

α is a preference parameter to balance the location proximity and term frequency. In Line 2, C is a priority queue that stores all the candidate terms. To get the candidate terms, we extract the summaries in root node of the quad-tree. However, if the candidates are stored in many nodes while the user specified k is only a very small number, the redundant computing of a large number of term scores will incur much additional time cost on calculating useless results. So we come up with a pruning method to avoid unnecessary computations while ensuring that we do not miss any 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 summaries are all sorted), we recompute their scores of "Distance part" as full values as their upper bound. When i-th ($i > k$) term has an upper bound score that is still smaller than the lower bound score of the k-th term, we believe that all the terms 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 Lines 4–13 in Algorithm 2.

Lines 4–13 show 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 Lines 12–13.

4 Experiments and Analysis

We conduct experiments to evaluate the solution and compare with other feasible methods. All the experiments are conducted on a workstation with Intel(R) Xeon(R) CPU $E5$-2643 0 @3.30 GHz 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, each of which 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.

4.1 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 the quad-tree. Specifically, in each leaf node of the quad-tree, we store the exact frequency of each term. When a message arrives, we update the frequency in 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 an LkTQ. Therefore, it can be used as a measure of querying accuracy in subsequent experiments.

4.2 Index Updating of Quad-Tree

First, we conduct an experiment that evaluates the performance when insert and remove a message in the sliding window. Since we aim to find the top-k term over a sliding window, when the sliding window is full, each time a new message is generated, an old message should be deleted.

We find that the two manipulations in baseline and our approach scarcely cost time as this is based on a well-constructed quad-tree. Therefore, we conduct another experiment to find out the time cost of constructing a quad-tree with all the term frequencies computed and index updating. Experiment results are shown in Fig. 3.

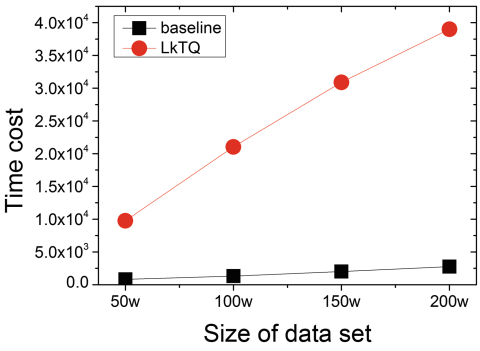


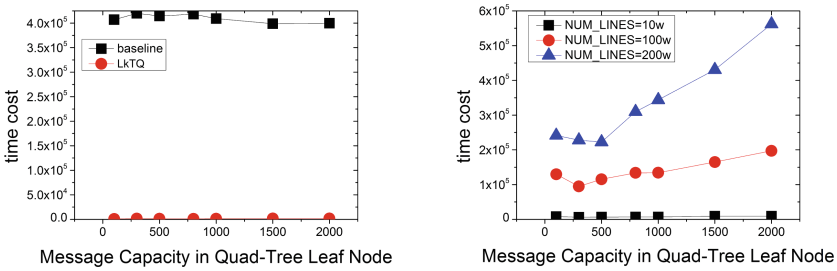
Fig. 3. Time cost of updating index on varied size of data

Specifically, for baseline, constructing the quad-tree includes counting and merging all the term frequencies while for our approach, the construction stage includes computing all the MG summaries of all the nodes in the quad-tree. As we can see, the time cost is much higher in our approach than in baseline. However, we conduct more experiments to prove that, even in this situation, our approach is much more efficient than the baseline.

4.3 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, then the node becomes a parent node and should 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.



(a) Time cost comparison (b) Time cost on varied message capacity

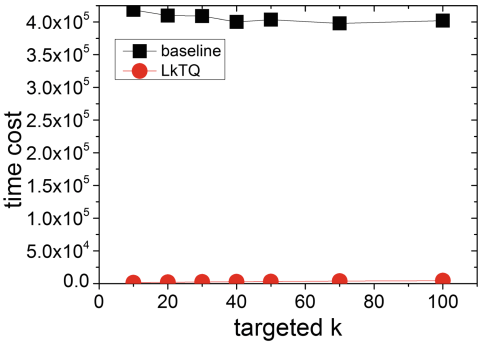
Fig. 4. Varying message capacity in quad-tree leaf node

Figure 4 shows the results. Figure 4a is the comparison results when the data set size is 10,000. M is ranged from 100 to 2000. Our approach is significantly faster than the baseline. 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 it fix M, the tree is fixed and the score is stable to be computed. 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, then the distance will be smaller and the number of leaf node is larger. As Fig. 4b shows, as M increases, time cost is higher. When M is getting larger, the cost of splitting is larger. There is a little turning down when M is around 300 and 500. In this range, it has almost the best performance.

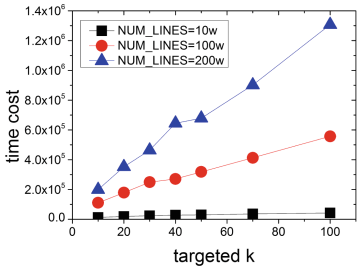
4.4 Varying Targeted k

In this experiment, we vary the targeted k. 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.

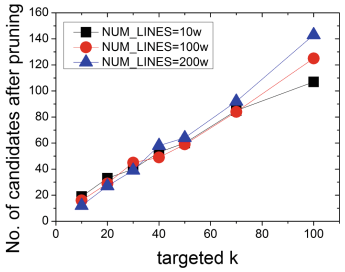
Figure 5 shows the results. The range of targeted k is set according to the normal requirements of users. The performance of our algorithm is remarkably better than the baseline, which counts one by one (Fig. 5a). The size of data set in Fig. 5a is 10,000, however, baseline need approximately seven minutes to return the results. The time cost of baseline is on a stable and inefficient level which is



(a) Time cost comparison for baseline and our algorithm



(b) Time cost on varied sizes of data



(c) No. candidates after pruning

Fig. 5. Varying targeted k

around 400,000 ms. For more larger data set, 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 Fig. 5b. Moreover, as the size 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 Fig. 5c as proof.

4.5 Accuracy Versus Baseline

Accuracy is a vital factor which users concern. The accuracy experiment results of our algorithm versus baseline are shown in Fig. 6. We measure the fraction of the correct top-k returning from our algorithm for different sizes 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.

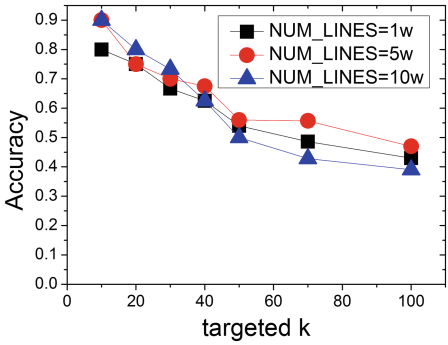


Fig. 6. Accuracy on varied sizes of data

4.6 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.

5 Conclusion

We propose a new approach for supporting querying the top-k locally popular and valuable 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 geo-textual social data streams over a sliding window and the framework is capable of returning results accurately and rapidly.

Acknowledgement. This work was supported by the NSFC (U1636210, 61373156, 91438121 and 61672351), the National Basic Research Program (973 Program, No. 2015CB352403), the National Key Research and Development Program of China (2016YFB0700502), the Scientific Innovation Act of STCSM (15JC1402400) and the Microsoft Research Asia.

References

1. Agarwal, P.K., Cormode, G., Huang, Z., Phillips, J., Wei, Z., Yi, K.: Mergeable summaries. In: PODS (2012)
2. Bansal, N., Koudas, N.: BlogScope: a system for online analysis of high volume text streams. In: VLDB (2007)
3. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Widmayer, P., Eidenbenz, S., Triguero, F., Morales, R., Conejo, R., Hennessy, M. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 693–703. Springer, Heidelberg (2002). doi:10.1007/3-540-45465-9_59
4. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. PVLDB **2**(1), 337–348 (2009)
5. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithms **55**(1), 58–75 (2005)
6. Cormode, G., Muthukrishnan, S.: What's hot and what's not: tracking most frequent items dynamically. TODS **30**(1), 249–278 (2005)
7. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: Möhring, R., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 348–360. Springer, Heidelberg (2002). doi:10.1007/3-540-45749-6_33
8. Felipe, I.D., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: ICDE (2008)
9. Finkel, R.A., Bentley, J.L.: Quad trees a data structure for retrieval on composite keys. Acta Inform. **4**(1), 1–9 (1974)
10. Li, F., Yao, B., Kumar, P.: Group enclosing queries. TKDE **23**(10), 1526–1540 (2011)
11. Li, F., Yao, B., Tang, M., Hadjieleftheriou, M.: Spatial approximate string search. TKDE **25**(6), 1394–1409 (2013)
12. Li, F., Yi, K., Tao, Y., Yao, B., Li, Y., Xie, D., Wang, M.: Exact and approximate flexible aggregate similarity search. VLDBJ **25**(3), 317–338 (2016)
13. Li, Y., Li, F., Yi, K., Yao, B., Wang, M.: Flexible aggregate similarity search. In: SIGMOD (2011)
14. Li, Z., Lee, K.C.K., Zheng, B., Lee, W., Lee, D.L., Wang, X.: IR-Tree: an efficient index for geographic document search. TKDE **23**(4), 585–599 (2011)
15. Lian, X., Chen, L.: Shooting top-k stars in uncertain databases. VLDBJ **20**(6), 819–840 (2011)
16. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. TODS **28**(1), 51–55 (2003)
17. Ozsoy, M.G., Onal, K.D., Altingovde, I.S.: Result diversification for tweet search. In: Benatallah, B., Bestavros, A., Manolopoulos, Y., Vakali, A., Zhang, Y. (eds.) WISE 2014. LNCS, vol. 8787, pp. 78–89. Springer, Cham (2014). doi:10.1007/978-3-319-11746-1_6
18. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In VLDB (2002)

19. Metwally, A., Agrawal, D., El Abbadi, A.: Efficient computation of frequent and top-k elements in data streams. In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, pp. 398–412. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30570-5_27
20. Metwally, A., Agrawal, D., El Abbadi, A.: An integrated efficient solution for computing frequent and top-k elements in data streams. TODS **31**(3), 1095–1133 (2006)
21. Misra, J., Gries, D.: Finding repeated elements. Sci. Comput. Program. **2**(2), 143–152 (1982)
22. Rocha-Junior, J.B., Gkorgkas, O., Jonassen, S., Nørvåg, K.: Efficient processing of top-k spatial keyword queries. In: Pfoser, D., Tao, Y., Mouratidis, K., Nascimento, M.A., Mokbel, M., Shekhar, S., Huang, Y. (eds.) SSTD 2011. LNCS, vol. 6849, pp. 205–222. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22922-0_13
23. Nutanong, S., Tanin, E., Zhang, R.: Incremental evaluation of visible nearest neighbor queries. TKDE **22**(5), 665–681 (2010)
24. Sankaranarayanan, J., Samet, H., Teitler, B.E., Lieberman, M.D., Sperling, J.: Twitterstand: news in tweets. In: GIS (2009)
25. Shang, S., Ding, R., Yuan, B., et al.: User oriented trajectory search for trip recommendation. In: EDBT (2012)
26. Shang, S., Ding, R., Zheng, K., et al.: Personalized trajectory matching in spatial networks. VLDBJ **23**(3), 449–468 (2014)
27. Shang, S., Zheng, K., Jensen, C.S., et al.: Discovery of path nearby clusters in spatial networks. TKDE **27**(6), 1505–1518 (2015)
28. Shang, S., Chen, L., Wei, Z., et al.: Collective travel planning in spatial networks. TKDE **28**(5), 1132–1146 (2016)
29. Shang, S., Chen, L., Jensen, C.S., et al.: Searching trajectories by regions of interest. TKDE **29**(7), 1549–1562 (2017)
30. Shang, S., Chen, L., Wei, Z., et al.: Trajectory similarity join in spatial networks. PVLDB **10**(11), 1178–1189 (2017)
31. Skovsgaard, A., Sidlauskas, D., Jensen, C.S.: Scalable top-k spatio-temporal term querying. In: ICDE (2014)
32. Teitler, B.E., Lieberman, M.D., Panozzo, D., Sankaranarayanan, J., Samet, H., Sperling, J.: Newsstand: a new view on news. In: GIS (2008)
33. Wang, Z., Wang, D., Yao, B., Guo, M.: Probabilistic range query over uncertain moving objects in constrained two-dimensional space. TKDE **27**(3), 866–879 (2015)
34. Xiao, X., Yao, B., Li, F.: Optimal location queries in road network databases. In: ICDE (2011)
35. Xie, D., Li, F., Yao, B., Li, G., Zhou, L., Guo, M.: Simba: efficient in-memory spatial analytics. In: SIGMOD (2016)
36. Xie, D., Li, G., Yao, B., Wei, X., Xiao, X., Gao, Y., Guo, M.: Practical private shortest path computation based on oblivious storage. In: ICDE (2016)
37. Yao, B., Li, F., Kumar, P.: Reverse furthest neighbors in spatial databases. In: ICDE (2009)
38. Yao, B., Li, F., Hadjieleftheriou, M., Hou, K.: Approximate string search in spatial databases. In: ICDE (2010)
39. Yao, B., Li, F., Kumar, P.: K nearest neighbor queries and KNN-joins in large relational databases (almost) for free. In: ICDE (2010)
40. Yao, B., Tang, M., Li, F.: Multi-approximate-keyword routing in GIS data. In: GIS (2011)
41. Yao, B., Li, F., Xiao, X.: Secure nearest neighbor revisited. In: ICDE (2013)

42. Yao, B., Xiao, X., Li, F., Wu, Y.: Dynamic monitoring of optimal locations in road network databases. VLDBJ **23**(5), 697–720 (2014)
43. Zhang, C., Zhang, Y., Zhang, W., Lin, X.: Inverted linear quadtree: Efficient top k spatial keyword search. In: ICDE (2013)
44. Zhang, D., Chan, C., Tan, K.: Processing spatial keyword query as a top-k aggregation query. In: SIGIR (2014)
45. Zhang, D., Tan, K., Tung, A.K.H.: Scalable top-k spatial keyword search. In: EDBT, pp. 359–370 (2013)
46. Zhao, K., Chen, L., Cong, G.: Topic exploration in spatio-temporal document collections. In: SIGMOD (2016)