2019 IEEE 35th International Conference on Data Engineering (ICDE)

Continuous Search on Dynamic Spatial Keyword Objects

Yuyang Dong*, Hanxiong Chen* and Hiroyuki Kitagawa *†
*Department of Computer Science, Information and systems, University of Tsukuba, Japan
†Center for Computational Sciences, University of Tsukuba, Japan
* touuyou@gmail.com, {*chx, †kitagawa }@cs.tsukuba.ac.jp

***Abstract*—As the popularity of SNS and the number of GPS-equipped mobile devices increases, a large number of web users frequently change their location (spatial attribute) and interesting keywords (keyword attribute) in real-time. An example of such would be when a user watches the news, videos, and blogs while moving. Many location-based web applications can benefit from continuously searching for these dynamic *spatial keyword objects*.**

In this paper, we define a novel query problem to continuously search for dynamic spatial keyword objects. To the best of our knowledge, this is the first work to consider dynamic spatial keyword objects. We employ a novel grid-based index to manage both queries and dynamic spatial keyword objects. With the proposed index, we develop a buffer named *partial cell list* to reduce the computation cost in the top-k reevaluation. The experiments confirm the superiorities of our proposed methods.

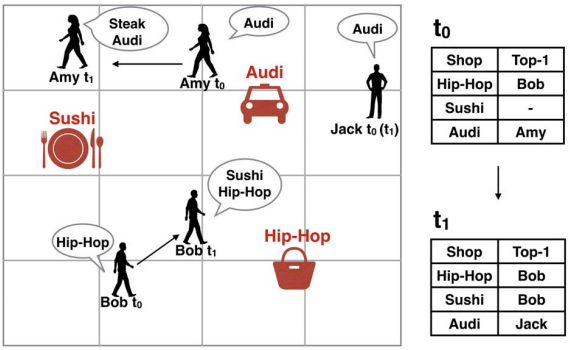


Fig. 1: E-coupon recommendation system.

I. Introduction

Smartphones and other mobile devices enable to receive information anywhere and enrich people's lives. People are used to watching the news, short video clips (e.g. Youtube, TikTok), posting on SNS (e.g. Twitter, Weibo) with smartphones while moving outside. Although different types of continuous spatial keyword queries have been studied [1]–[4], the existing research considers only static objects. In this research, we define a novel searching problem that continuously searches for top-k dynamic spatial keyword objects.

Figure 1 shows a scenario that explains how our research works with the application of the E-coupon recommendation system. Assume that a Hip-Hop cloth store, a Sushi restaurant and an Audi car dealer are registered on our system. Our system continuously searches the top-1 people for these three shops and sends out coupons. At a certain time t_0, Bob is watching a hip-hop music video on his cellphone and becomes the top-1 result of the Hip-hop store. Amy searches "Audi car" with her phone, and Jack is watching a news about "Audi car". Our system adds Amy to the top-1 for the Audi dealer since she is closer than Jack. Nobody is watching content about "Sushi", so the top-1 for the Sushi restaurant is empty. At t_1, Bob has changed his location and has started to watch an eating show about Sushi. Then the keyword attribute of Bob changes to "Sushi, Hip-Hop" [1]. Our system still keeps Bob as the top-1 of the Hip-Hop shop since there are no other better options, and adds Bob to the top-1 list of the Sushi restaurant. Amy has left away from the Audi dealer and has searched "Steak near me" on her cellphone. Our system recognizes Amy's change and has reevaluated the top-1 for Audi dealer, finding that Jack who has stayed and still watching the news about the "Audi car" becomes the top-1.

[1]We suppose that the keyword attribute remains some keywords of the previous status.

II. Preliminaries

Definition 1: (***Dynamic Spatial Keyword Object,*** o). A dynamic spatial keyword object o is defined as $o = (o.\rho, o.\psi, t)$, where $o.\rho$ is the location attribute with coordinates, $o.\psi$ is a set of keywords, and t is the timestamp. Both $o.\rho$ and $o.\psi$ change over time. $o.\rho$ is updated with the up-to-date location. $o.\psi$ keeps the keywords of the up-to-date m status of object o, where m is a user-determined window size.

Definition 2: (***Spatial Keyword Query,*** q). Spatial keyword query q also has a location attribute and a set of keywords, $q.\rho$ and $q.\psi$. In addition, $q.k$ is the number of results (the k in top-k). $q.\alpha$ is a user-defined smoothing parameter for spatial keyword similarity. The attributes of a query are static.

Definition 3: (***Spatial Keyword Similarity,*** $SimST$). Given object o and query q, the spatial keyword similarity between them is defined as:

$$SimST(o,q) = q.\alpha \cdot SimS(o.\rho, q.\rho) + (1 - q.\alpha) \cdot SimT(o.\psi, q.\psi) \quad (1)$$

$SimST$ is a combined value of spatial similarity $SimS$ and keyword similarity $SimT$. Firstly, the $SimS$ is calculated by the normalized Euclidean similarity. Note that the $maxDist$

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00146

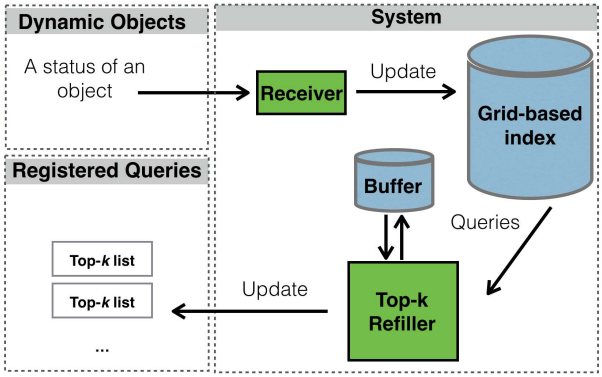


Fig. 2: The system and flow of the process: *Grid-based index*, *Top-k refiller* and *PCL buffer*.

is the maximum distance in the data space.

$$SimS(o.\rho, q.\rho) = 1 - \frac{Euclidean(o.\rho, q.\rho)}{maxDist} \quad (2)$$

Secondly, $SimT$ is computed by the inner product between the tf-idf weights of $q.\psi$ and $o.\psi$.

$$SimT(o.\psi, q.\psi) = \sum_{w \in o.\psi \cap q.\psi} wt(o.w) \cdot wt(q.w) \quad (3)$$

where $wt(w)$ denotes the tf-idf weight of keyword w, and the weights of objects and queries are normalized to the unit length.

Definition 4: (***Continuous Search***). Given an object set O and a query set Q, for each query $q \in Q$, the continuous search is to keep the up-to-date top-k objects o's ($o's \in O$) ranked by descending order of $SimST(o, q)$.

Figure 2 is an overview of how the proposed system solves the continuous search on dynamic spatial keyword objects.

III. Grid-based index

We use a regular grid-based index to maintain both objects and queries (but they are not in the same indexing rule) because the data in the grid can be accessed and updated directly.

Objects are indexed in the grid w.r.t. their located cells. On the other hand, to find the affected queries efficiently for a new status of an object, the queries are indexed into the cell according to their "influential circles". We use $q.\rho$ as the center point and create an influential circle with a radius r calculated by:

$$r = \frac{1 - kScore(q)}{q.\alpha} \cdot maxDist \quad (4)$$

$kScore(q)$ is the smallest score in the top-k list of q. If an object is located outside of this influential circle, it will not be an element in the top-k of q. We indexed q into the cells that overlap q's influential circle. Figure 3 shows an example where the circle of q_1 overlaps with c_1, c_2, c_3, and c_4. Hence, q_1 is indexed into these four cells.

A cell can bound the range of the spatial attribute for the objects. For the keyword attribute, maximum weights (*maxwts*) and minimum weights (*minwts*) of the objects in a cell are also indexed. The *maxwts* (*minwts*) are used to bound keyword similarities between a query and a cell.

IV. Proposed Methods

When a dynamic object o changes to o', there are two kinds of affected queries, OutQ and InQ. OutQ is a set of queries corresponding to the previous o, each query in OutQ contains o in its top-k. InQ is a set of queries corresponding to the current o', each query in InQ has a larger $SimST(o', q)$ than $kScore(q)$. Usually, OutQ is initialized when the system starts and is recorded in the object table. InQ can be retrieved from the indexed queries of the located cell of o'.

Updating the top-k's for queries of InQ is low-cost since only the top-k list is considered with o'. In other words, we just need to insert o' into the previous top-k. However, for the queries of OutQ, o' may get out of the top-k list. In this case, we must reevaluate the top-k result from all objects. It is natural that the cardinality of the objects is always much larger than k. Consequently, compared to the process of InQ, the main task is to reevaluate the OutQ's top-k lists efficiently.

A. GCL *method with the sorted cell list*

The related research focuses on maintaining the candidate objects in a result buffer to support the top-k reevaluation. However, keeping objects in the buffer is inefficient in our problem because the objects are dynamic, that incurring frequent and expensive buffer maintenance. To address this limitation, we propose a sorted cell list (CL) buffer that maintains all cells of our grid-based index w.r.t. a similarity priority.

We use *q.CL* to denote the sorted cell list w.r.t the query q. All cells in the grid-based index are sorted by their $maxscore$ and stored in CL, where $maxscore(c, q)$ is the upper value of the spatial keyword similarity between any object in cell c and a query q:

$$maxscore(c, q) = q.\alpha \cdot SimSUB(c, q.\rho) + (1 - q.\alpha) \cdot SimT(c.maxwts, q.\psi) \quad (5)$$

Our idea is that we can employ an efficient branch-and-bound method to find the top-k objects from the cells in CL. Since CL can be implemented as a binary-tree-like structure [2].

B. GPCL *method with the partial sorted cell list*

GCL has a limitation that the buffer CL must maintain itself every time even though an object does not affect any queries. Motivated by the above limitations, we propose a partial sorted cell list (PCL), which is a subset of CL. PCL always keeps the candidate $(k+1)$-th object to refill the top-k list.

Besides the $maxscore$ in Equation (5), $minscore$ of a cell is also indexed in PCL. The $minscore(c, q)$ denotes the minimum spatial keyword score between any object in cell c and a query q. $minscore$ is formed with the minimum spatial

[2]Our implementation uses std::map in C++, it is a red-black tree structure.

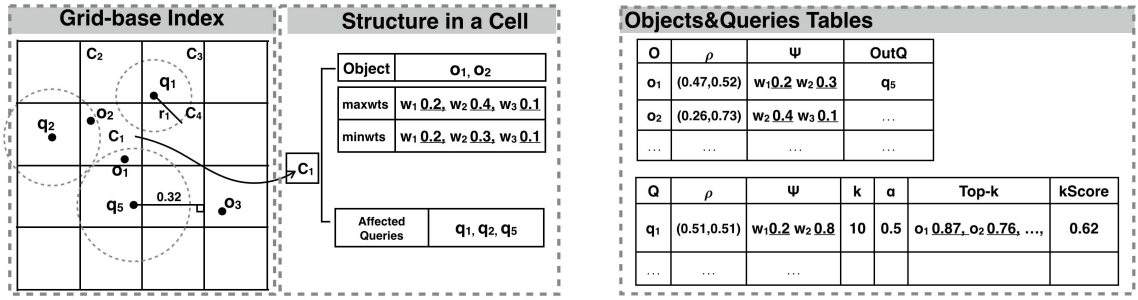


Fig. 3: Grid-based index, inner structure of a cell, and tables

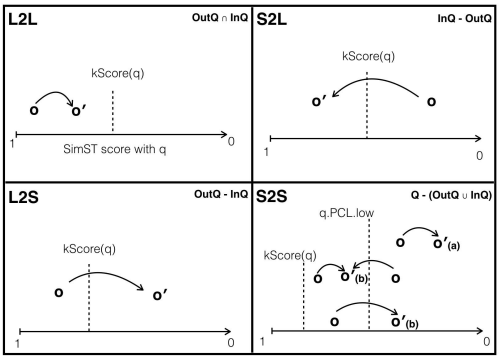


Fig. 4: Four cases of a dynamic object and a query.

similarity ($SimSLB$) and the minimum keyword similarity ($SimTLB$).

$$minscore(c, q) = \alpha \cdot SimSLB(c, q.\rho) + (1 - \alpha) \cdot SimTLB(c.minwts, q.\psi) \quad (6)$$

$$SimTLB(c.minwts, q.\rho) = MIN_{w \in q.\psi \cap c.minwts} wt(q.w) \cdot wt(c.w) \quad (7)$$

1) **Partial sorted cell list (PCL):**

Definition 5: (*Partial Cell List, PCL*). Given a query q and a grid-based index. For each cell $c \in q.PCL$, it holds that c contains at least one object, and $minscore(c, q) < q.PCL.up$ and $maxscore(c, q) > q.PCL.low$. While processing, $q.PCL.up$ is updated as the value of up-to-date $kScore(q)$, $q.PCL.low$ is initialized as the value of $maxMinS$ and will not change until $q.PCL$ is recreated, where $maxMinS = MAX\{minscore(c_j, q)\}$, $c_j \in grid.cells$ and $maxscore(c_j, q) < kScore(q)$.

For each query, the corresponding PCL is initialized based on Definition 5. When a top-k list needs to refill, a candidate object can be searched from PCL and refilled to this top-k list. The top-k reevaluation is much more efficient than using the CL because only a top-1 search is conducted from fewer cells. To guarantee that PCL always contains a candidate object to refill top-k, we propose a sophisticated strategy to maintain PCL.

2) **PCL Maintenance:** We divide the situations between a dynamic object and a query into the following four cases. Figure 4 shows images and summarizes the four cases: **L2L** (large to large), **S2L** (small to large), **L2S** (large to small) and **S2S** (small to small). Because a dynamic object may affect the cells in PCL, we should maintain PCL carefully to ensure it always contains the (k+1)-th object to refill. We propose a sophisticated maintenance process.

L2L and S2L. The cases of L2L and S2L are discussed together since they have a common PCL maintenance. In the cases of L2L, both o and o' are in the top-k. Thus, the order of the objects outside top-k is not changed, and the $(k+1)$-th object remains in PCL. Therefore, we just check the changing cells $o.cell$ and $o'.cell$ with PCL. For the case of S2L, o' is added into top-k from the outside. The previous kth object (denoted as o_k) will become the $(k+1)$-th one, so we should add the $o_k.cell$ into PCL to ensure the candidate object. $o.cell$ and $o'.cell$ also require to check.

L2S. In the situation of L2S, we must search the top-1 object o_{cand} from PCL. Then we compare $SimST(o', q)$ with $SimST(o_{cand}, q)$ and determine whether to refill o_{cand} or not. PCL is trimmed with a new score range $(kScore(q)', q.PCL.low)$ where $kScore(q)'$ denotes the updated k-th score.

S2S. S2S is divided into two sub-cases: S2S.a and S2S.b. In S2S.a, both o and o' are outside of $q.PCL.low$, so PCL does not need to be maintained. In S2S.b, we need to check $o.cell$ and $o'.cell$ with PCL. Similar to L2S, we will recreate PCL if it becomes empty.

Algorithm 1 gives the proposed GPCL method.

V. Experiments

All algorithms were implemented in C++. All indices, buffers, and algorithms were run on in-memory of a Mac with a 2.2GHz Intel Core i7 CPU and 32GB memory. We used two real datasets and one synthetic dataset. **YELP** is an open source dataset provided by YELP.com[3]. **TWITTER** is the dataset with 4.2M geo-tag tweets from the United States[4]. In TWITTER, there are 1.2M unique users each of which has at least three geo-tag tweets. **SYN** is a synthetic data containing 12M spatial keyword tuples. Spatial attributes are generated

[3]https://www.yelp.com/dataset

[4]https://datorium.gesis.org/xmlui/handle/10.7802/1166

Algorithm 1 GPCL

Input: o, o', $OutQ$, InQ

// **L2L**
for each $q \in OutQ \cap InQ$ **do**
  Update $top\text{-}k(q)$ with o'.
  $q.PCL.check(\{o.cell \cup o'.cell\})$
// **S2L**
for each $q \in InQ - OutQ$ **do**
  Update $top\text{-}k(q)$ with o'.
  $q.PCL.check(\{o.cell \cup o'.cell \cup k.cell\})$
// **L2S**
for each $q \in OutQ - InQ$ **do**
  o_{cand} = Retrieve Top-1 from $q.PCL$
  if $SimST(o', q) > SimST(o_{cand}, q)$ **then**
    Update $top\text{-}k(q)$ with o'.
  else
    Update $top\text{-}k(q)$ with o_{cand}.
  $q.PCL.check(\{o.cell \cup o'.cell\})$
  $q.PCL.trim(kScore(q)', q.PCL.low)$
  if $q.PCL$ is empty **then**
    $q.PCL.recreate$
// **S2S**
for each $q_i \in Q - OutQ \cup InQ$ **do**
  if $SimST(o, q) < q.PCL.low$ and $SimST(o', q) < q.PCL.low$ **then**
    continue
  else
    $q.PCL.check(\{o.cell \cup o'.cell\})$
    if $q.PCL$ is empty **then**
      $q.PCL.recreate$

by the BerlinMOD benchmark [5]. Keyword attributes are used from the keywords in TWITTER. We compare the following methods:

- **CIQ-kmax.** The block-based inverted file structure in [4] with the $kmax$ buffer in [5].
- **IGPT-kmax.** The group pruning techniques in [3] with the $kmax$ buffer in [5].
- **AQF-GCL.** The proposed method with CL.
- **AQF-GPCL.** The proposed method with PCL.

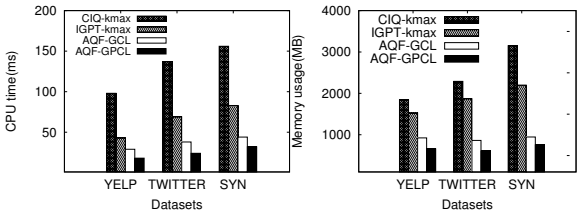
Overall processing. Figure 5 shows the comparison results for the overall processing. Both of our proposed methods, AQF-GCL and AQF-GPCL, have better performances than the others on both processing time and memory cost.

Effect on varying k. According to Figure 6a, AQF-GPCL is the best method and is not influenced by k. From the memory usage of indices in Figure 6b, a large k leads to a bigger index for $kmax$-based methods. Our proposed methods are unaffected by k since we index the identity of the cells.

VI. Conclusion

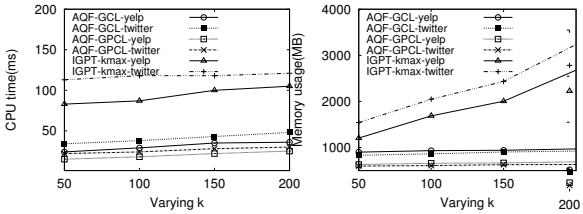
In this paper, we investigate a novel problem that searches dynamic spatial keyword objects continuously and propose a solution system. We employ a grid-based index to handle both dynamic objects and queries. We propose a sophisticated buffer called partial cell list (PCL) to efficiently refill the top-k results in our *top-k refiller* module. The experiments confirm that our proposal has a good performance compared with the baselines and related works.

[5]http://dna.fernuni-hagen.de/secondo/BerlinMOD/BerlinMOD.html



(a) Average processing time. (b) Memory usage.

Fig. 5: Overall processing.



(a) Average processing time. (b) Memory usage.

Fig. 6: Varying k.

Acknowledgement

This research was partly supported by the program "Research and Development on Real World Big Data Integration and Analysis" of RIKEN, Japan.

References

[1] D. Wu, M. L. Yiu, and C. S. Jensen, "Moving spatial keyword queries: Formulation, methods, and analysis," *ACM Trans. Database Syst.*, vol. 38, no. 1, pp. 7:1–7:47, 2013. http://doi.acm.org/10.1145/2445583.2445590

[2] B. Zheng, K. Zheng, X. Xiao, H. Su, H. Yin, X. Zhou, and G. Li, "Keyword-aware continuous knn query on road networks," in *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, 2016, pp. 871–882. https://doi.org/10.1109/ICDE.2016.7498297

[3] X. Wang, Y. Zhang, W. Zhang, X. Lin, and Z. Huang, "SKYPE: top-k spatial-keyword publish/subscribe over sliding window," *PVLDB* no. 7, pp. 588–599, 2016. http://www.vldb.org/pvldb/vol9/p588-wang.pdf vol. 9,

[4] L. Chen, G. Cong, X. Cao, and K. Tan, "Temporal spatial-keyword top-k publish/subscribe," in *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015* pp. 255–266. https://doi.org/10.1109/ICDE.2015.7113289 , 2015,

[5] K. Yi, H. Yu, J. Yang, G. Xia, and Y. Chen, "Efficient maintenance of materialized top-k views," in *Proceedings of the 19th International Conference on Data Engineering, ICDE, March 5-8, 2003, Bangalore, India*, 2003, pp. 189–200. https://doi.org/10.1109/ICDE.2003.1260792

[6] K. Mouratidis, S. Bakiras, and D. Papadias, "Continuous monitoring of top-k queries over sliding windows," in *Proceedings of the ACM SIGMOD International Conference on Management of Data ,Chicago, Illinois, USA, June 27-29, 2006*, 2006, pp. 635–646. http://doi.acm.org/10.1145/1142473.1142544