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 the SNS and GPS-equipped mobile devices increases, a massive of web users frequently change their location (spatial attribute) and profile (keyword attribute) in real-time such as a user tweets his feeling while traveling. Many location-based web applications can benefit from continuously searching these dynamic *spatial keyword object*. For example, a real-time coupon delivery system can search people by matching their locations and profiles, then send coupons to attract customers.**

In this paper, we define a novel query problem to continuously search the dynamic spatial keyword objects. To the best of our knowledge, this is the first work to concern the 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 devise a group-pruning technique to efficiently find the queries affected by the dynamic objects. Moreover, to quickly update the searching results for affected queries, we develop a buffer named *partial cell list* to reduce the computation cost in the top-k reevaluation. Theoretical analysis and experiments confirm the superiorities of our proposed techniques.

I. Introduction

Nowadays, people are more likely to access the web from mobile devices such as a smartphone or a tablet than a desktop or a laptop computer. The prevalence of GPS-equipped mobile devices makes massive volumes of dynamic spatial keyword data generated and updated on the web, such as people post on SNS, micro-blogs and crowd-sourced reviews with the geo-tags. In the real world, a spatial keyword object is a dynamic data such as a man change his preferred food (keyword attribute) while walking around (location attribute). Many applications can benefit from continuous searching on these dynamic spatial keyword data. An example is the location-awareness recommendation system. For instance, a business owner (e.g., a restaurant or a shop) can explore potential customers by continuous searching target people with location and keyword (profile contents) attributes. Besides the touristic advertising scenario, another application is the dynamic object tracking. For example, a rescue center can monitor people's health condition by searching the information (keyword attribute) and location (spatial attribute) which updated by the wearable devices (e.g., an Apple Watch).

Although different types of continuous spatial keyword queries have been studied, the existing research only consider the static objects. In other words, there is no research on dynamic spatial keyword problem, which is more realistic in the real world applications. [1]–[4] consider applications that maintain top-k static objects (e.g., gas-stations) for a dynamic query (e.g., a vehicle in motion). This condition differs from our problem, we aim to maintain top-k dynamic objects for static queries. From another perspective, researchers have investigated streaming Publish/Subscribe systems, such as CIQ [5] and SKYPE [6]. These studies differ from the problem in our research because they only consider the incremental static objects. In these works, the top-k results are updated with the new coming objects but these objects will not change. Using tweets searching as an example, both CIQ and SKYPE search the top-k tweets while new tweets are posting. In contrast, we search top-k users while the users are updated with their recent tweets. Actually, if we have to achieve searching dynamic objects in a streaming manner, we could treat an update operation as a combination of a deletion operation and an insertion operation. Unfortunately, existing research cannot deal with a random pairwise insertion/deletion (i.e., an unpredictable change of an object). Because CIQ is an append-only system without deletions and SKYPE is based on the sliding window, insertions and deletions must be orderly w.r.t the sequence of streaming data.

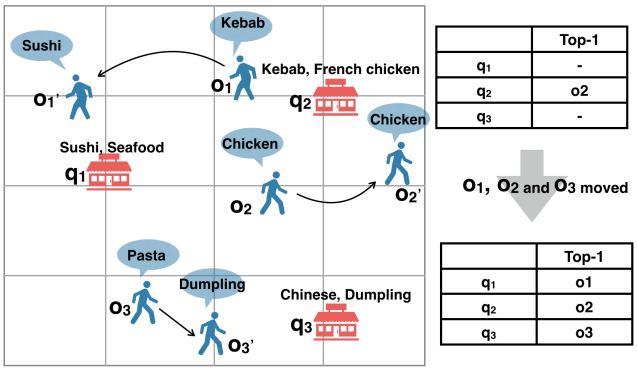


Fig. 1: E-coupon recommendation system.

This gap in the existing research has motivated us to study the realistic problem of continuous search on dynamic spatial keyword objects.

A. Example

We use a toy example as an application to explain our problem. Figure 1 shows a simple E-coupon recommendation system. There are three people ($o_1 - o_3$) and three restaurants ($q_1 - q_3$). On the right side, the two tables describe the top-1 results for each restaurant before and after the changes of the

objects. Note that o_1' means the status of o_1 after change. As o_1 moves, his keyword also changes from "kebab" to "sushi". Restaurant q_1 detects o_1 as a matching customer and updates its top-1 list. Then the system delivers a coupon to attract o_1 as a customer for q_1. Restaurant q_2 keeps the same result since o_2 remains as the top-1. Finally, o_3 gets close to q_3 and becomes to contain the same keyword "dumpling" of q_3. Hence, the system adds o_3 into the q_3's top-1 result.

B. Challenges and contributions

Challenges. There are two challenges in our research.

1. Indexing objects and queries. The first challenge is to design efficient indexing structures to manage the big data of dynamic objects and queries. The most relevant works of index streaming spatial keyword objects are the Publish/Subscribe systems CIQ [5] and SKYPE [6] we mentioned before. We first discuss the object index. CIQ and SKYPE used an IR-tree structure [7] to index the objects since they do not update the objects in the index. The IR-tree (Inverted File R-tree) has a similar structure as the R-tree, and each node links an inverted file to index the keyword information. Therefore, an IR-tree suffers from the well-known limitation [8] of an R-tree where structure updates have an expensive cost. We can not use the existing spatial keyword indexing technique for the dynamic objects.

Secondly, when receive a new status of a dynamic object, we need to find the affected queries promptly. Both CIQ and SKYPE use a quad-tree structure to index queries. CIQ sets a decay function according to the similarity score, so all queries must be indexed into every leaf-node. This feature looses the spatial pruning, and leads to an extremely high memory cost. SKYPE sets a non-decaying similarity function so that a single query is indexed into one leaf-node. However, it still requires overhead computing to target affected queries by traversing the tree. Consequently, it is a challenge to index queries and to retrieve them efficiently.

2. Top-k reevaluation. Another critical challenge is the top-k reevaluation, which occurs after a dynamic object changes out of the top-k list. In this situation, it is time-consuming if the top-k is naively reevaluated from the scratch of all objects. It is also infeasible to buffer all objects with their scores for each individual query to avoid the top-k reevaluation. Because the maintenance cost is extremely high for the dynamic objects, and the tremendous cardinality of the queries incurs expensive memory consumption. Therefore, related techniques have been proposed to balance the trade-off between the number of reevaluations and the buffer size. Yi et al. [9] introduced a $kmax$ buffer, which maintains top-k' results, and k' is a value between k and $kmax$. However, the $kmax$ solution just transforms the top-k maintenance to top-$kmax$ maintenance. The problem on how to reduce the number of evaluations remains unsolved. Later, Wang et al. [6] study the top-k maintenance with spatial keyword objects and develop a cost-based k-skyband strategy to select a proper threshold θ via theoretical analysis of the cost-model. Unfortunately, the above k-skyband framework is proposed for the streaming process with a sliding window. Specifically, the k-skyband buffer is constructed and maintained with the order of the incoming objects sequence, which is inapplicable to a scenario of searching dynamic objects because the dynamic objects always change randomly and unpredictably. Hence, the design of an appropriate buffer with theoretical underpinnings to efficiently process the top-k reevaluation remains a serious challenge.

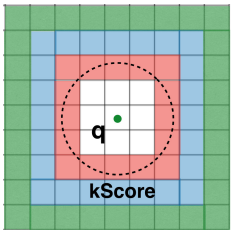


Fig. 2: Priorities of cells with spatial-only similarity.

Many research on moving objects [10]–[13] utilize a grid-based index to manage the moving objects. Because they only consider the dynamic of spatial attribute (i.e., moving), they can take advantages of the priority features among cells and search the candidate objects efficiently. Figure 2 shows the priorities of cells. To find a nearest object (the $k+1$ object) beyond the top-k, we can find the red cells first, then if there exist at at least one object which not in the top-k, we just only check the blue cells and the green cells can be safely skipped. However, these priority constraint can not work with the spatial keyword similarity of our research. For example, there may exist an object in the green cells but have a higher relevance of keyword similarity, that makes it become the most similar one than any objects in the red and blue cells. In conclusion, the technique of moving object can not be used with the dynamic spatial keyword object.

Contribution. In this paper, we define a new query process, which continuously searches the top-k dynamic spatial keyword objects. To overcome the first challenge of indexing, we design a novel grid-based index to manage both dynamic objects and static queries. The grid-based index can support rapid and economical updates of dynamic objects. In addition, queries are indexed with a sophisticated strategy of influential circle, and queries affected by adynamic object can be quickly identified. For the second challenge of the top-k reevaluation, we proposed a novel strategy that refills one candidate object rather than reevaluates the entire top-k list. Taking advantage of the cells in the grid-based index, we design a buffer named PCL (partial cell list). PCL balances the trade-off between the search process and buffer maintenance to optimize efficiency. Our principal contributions are:

- We formalize the continuous search problem on dynamic spatial keyword objects.
- We designed a grid-based index to manage both objects and queries. Employing the cells in the grid-based index, we propose sophisticated strategies on affected queries finding and top-k refilling. In top-k refilling, we propose a buffer named PCL, which has a theoretical basis to maximize the efficiency of the continuous searching process.
- We conducted extensive experiments to verify the efficiency and effectiveness of our proposed methods.

II. Related Work

Searching spatial-only moving objects. Our work is related to the problem of continuously searching spatial k nearest neighbor (kNN) queries over moving objects. This kind of research [10]–[14] aims to keep the kNN moving objects to a fixed location (query point). Mouratidis et al. [10] proposed a grid index and the concept of the influence region which enlighten us to design a grid-based index for both dynamic spatial keyword objects and queries. However, the above solutions do not consider the keyword similarity so these techniques can not be extended to our research.

Snapshot spatial keyword search. Research on searching geo-textual objects with query locations and keywords are widely studied. Various works have retrieved spatial keyword objects by different type of queries such as boolean matching [7] or a combined score function evaluation [15]. The survey paper [16] and tutorials [17], [18] give sufficient summaries of different problem settings and techniques in the spatial keyword search. Note that these works provide a snapshot query for static datasets, whereas our problem focuses on a continuous query with dynamic spatial keyword data.

"Static objects, moving query". An example of a "static objects, moving query" is keeping top-k gas-stations for a driving car. Literally, in this continuous query, the query is moving but the objects are unaltered. Wu et al. first proposed a continuously moving top-k spatial keyword (MkSK) query [1], [2] using a filtering technique of safe-region on multiplicatively weighted Voronoi cells. Huang et al. [3] pointed out that the ranking function in [1] is ad-hoc. Hence, they studied MkSK queries with a general weighted sum ranking function and proposed a hyperbola-based safe-region to filter objects. In the above research, spatial similarity is evaluated using the Euclidean distance. On the other hand, Zheng et al. [4] studied a continuous boolean top-k spatial keyword query over a road network with the techniques of the graph. Guo et al. [19] studied a continuous top-k spatial keyword query with a combined ranking function. Obviously, these works differ from ours since we focus on the problem of "dynamic objects, static query".

Publish/Subscribe system. Another type of research of continuous search on spatial keyword objects is the Publish/Subscribe system. Users register their interest as a continuous query into the Publish/Subscribe system, then new streaming objects are delivered to relevant users. Similar to a snapshot spatial keyword search, the research topic is also separated by the matching function. Boolean matching is studied in [20]–[22], while the combined value between spatial similarity and keyword relevance are considered in [5], [6]. Note that these works are inherently different from ours problem setting since they do not consider the dynamic objects. Moreover, as we mentioned in Section I, the techniques used in these works can not deal with the random pairwise insertion/deletion of objects. Hence, it is impossible to apply these techniques into our problem with dynamic spatial keyword objects.

Table I compares our work to existing works.

Research	Object	Query	Attribute(s)
[10]–[14]	moving	static	spatial
[16]–[18]	static	static	spatial keyword
[1]–[4], [19]	static	moving	spatial keyword
[5], [6]	streaming	streaming	spatial keyword
Our work	dynamic	static	spatial keyword

TABLE I: Comparison of our work to existing works.

Symbols	Description
o, O	object, objects set
q, Q	query, query set
$top\text{-}k(q)$	k objects with the largest scores with q
$kScore(q)$	smallest score in the $top\text{-}k(q)$
o, o'	previous status and current status of a dynamic object
$grid$	grid-based index
c	a cell in the grid-based index.
n, n^2	partition number in a grid, cells number in a grid
$o.cell$	cell in a grid where object o is located
OutQ	q's which contain o in their top-k
InQ	q's satisfying $SimST(o', q) > kScore(q)$
$maxscore(c, q)$	maximum score of cell c and q
$minscore(c, q)$	minimum score of cell c and q
CL	sorted cell list
PCL	partial sorted cell list

TABLE II: Notations and symbols

III. Preliminaries

A. Definitions

In this section, we formally define the dynamic spatial keyword object, the spatial keyword query, the score function between them, and the problem continuous search on them. Table II summarizes the notations frequently used in this paper.

Definition 1: (***Dynamic Spatial Keyword Object,*** o). A dynamic spatial keyword object o is defined as $o = (o.\rho, o.\psi)$, where $o.\rho$ is the location attribute with the coordinates, and $o.\psi$ is a set of keywords. Both $o.\rho$ and $o.\psi$ change over time.

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

Hereafter, we abbreviate dynamic spatial keyword object as *object* and spatial keyword query as *query*. It is important to note that both location attribute and keyword attributes of an object are changing over time. Although a query is defined as static, we also allow incremental queries. Because it is meaningless to change the query conditions. However, it makes sense to add new queries with different targets. To evaluate the relevance of an object o to a query q, we define a score function as follows:

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

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

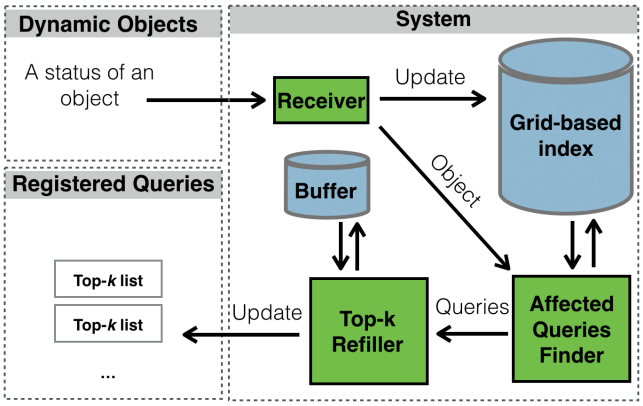


Fig. 3: System and flow of the monitoring process.

$SimST$ 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 $maxDist$ in the equation 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 weights vector of keyword w, and the weights of objects and queries are normalized to the unit length. The functions to find the similarities are also used in the related work [6]. To ensure that both spatial and keywords are relevant, we require that every object in the top-k of a query must contain at least one common keyword with this query.

Finally, we define the problem of the continuous search with above objects and queries.

Definition 4: Given an object set O and a query set Q, for each query $q \in Q$ the continuous search is to keep the current top-k objects o's ($o's \in O$) ranked by the descending order of $SimST(o, q)$. [1]

B. Proposed System Overview

Figure 3 overviews the propose system to solve the continuously search on dynamic objects. We assume that at the initial state of the system, there already exists some objects and queries, as well as the top-k results. The new coming status of dynamic objects are handled sequentially while processing. When receiving a new status of an object, the grid-based index is updated first (Section IV). Then the **affected queries finder** is executed to find queries affected by this object (Section V). Finally, the top-k lists of these queries are updated. If a dynamic object causes the result to become short of k-elements, then the **top-k refiller** is triggered to refill the top-k list by checking it against a result buffer of candidates (Sections VI and VII). Since the data can be easily inserted and removed from the grid-based index, our system also supports the incremental (decremental) of objects and queries.

[1]When the similarities between two objects are equal, we assume that either of two objects can be correct for the result.

IV. Grid-based Index

Since the objects are dynamic data, the index should respond quickly and have a low updating cost. Similar to the existing research [10]–[13], 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 (o(1) complexity). Unlike some complicated indices (R-tree, IR-tree, etc.) that require extra and expensive costs to maintain the structure, the grid-based index is better suited for frequently updated applications.

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

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

where $maxDist$ is the maximum distance in the space and $kScore(q)$ 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 4 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 are also indexed in a cell. The *maxwts* (*minwts*) can bound keyword similarities between a query and a cell, the details is in Section V and Section VI.

Example 1: In Figure 4, cell $c1$ contains objects o_1 and o_2. According to the informations of o_1 and o_2 in the Object table, $c1.maxwts = \{0.2, 0.4, 0.1\}$ and $c1.minwts = \{0.2, 0.3, 0.1\}$, corresponding to the keywords w_1, w_2 and w_3.

In summary, a grid cell contains four kinds of information: objects, $maxwts$, $minwts$ and affected queries. Note that the objects are indexed into a sequential list, but the affected queries are indexed into a sophisticated group-based structure. This arrangement supports efficient group-pruning technique used in our *Affected Queries Finder* module. (Section V). The information and useful statistics of objects and queries are stored in two different tables, we only index the object id and query id into the grid-based index. Since our system is an in-memory system, all other information can be retrieved from the tables via the random access.

V. Affected Queries Finder

In this section, we introduce the *Affected Queries Finder* (AQF) module. When the system receives a new status of a dynamic object, AQF helps to find the affected queries ,i.e., the queries whose top-k needs to updated. We use o and o' to represent the previous status and the current status,

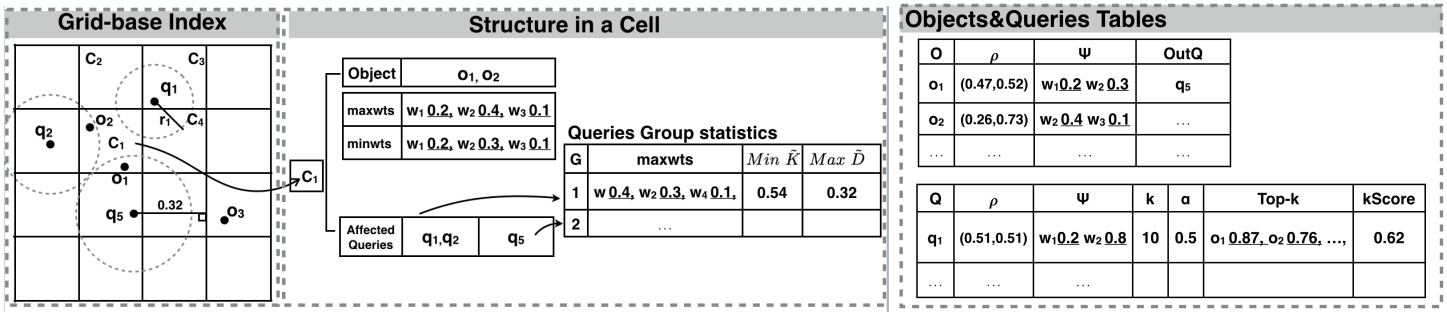


Fig. 4: Grid-based index, inner structure of a cell, and tables

respectively. Updating from o to o' affects two kinds of 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. (i.e., the dynamic o' may reduce OutQ's top-k lists to less than k objects). On the other hand, when object o change to o', InQ collects queries corresponding the current o' that $SimST(o', q)$ becomes larger than their $kScore$. (i.e., the dynamic o' make InQ's top-k lists updated with o').

$$InQ = \{q \mid q \in Q \wedge SimST(o', q) > kScore(q))\} \quad (5)$$

Usually, OutQ is initialized when the system starts, and is updated in the object table when a process for a dynamic object is completed. Therefore, we can retrieve OutQ easily by looking up the object table, and the problem of identifying affected queries becomes how to efficiently find InQ. In the rest of this section, we introduce the techniques of finding InQ.

Influential circle pruning. If a dynamic objects locate much too far from a query q, then q can be safely pruned. This is the reason that we index a query into the overlapping cells w.r.t its influential circle, and only the indexed queries in the cell $o'.cell$ need to be considered.

Group query pruning. To avoid compare the indexed queries with o' one by one, for each cell, we divide the indexed queries into several groups. Then a threshold for each group is generated to prune multiple queries simultaneously. Let see the threshold for a single query first.

Single threshold. Assuming that an object moves to a cell (denote as $o'.cell$), then for an arbitrary query q, the upper bound of the spatial similarity to q ($SimSUB(o'.cell, q)$) can be estimated based on Equation (2):

$$SimSUB(o'.cell, q) = 1 - \frac{minD(o'.cell, q.\rho)}{maxDist} \quad (6)$$

where $minD(o'.cell, q)$ is the minimum Euclidean distance between q and $o'.cell$ (e.g. in Figure 4, $minD(o_3.cell, q_5) = 0.32$. Note that if q is covered by $o'.cell$ then $minD(o'.cell, q) = 0$). We can infer a keyword similarity threshold $T_\theta(o'.cell, q)$ with $SimSUB(o'.cell, q)$ and Equation (1),

$$T_\theta(o'.cell, q) = \frac{kScore(q)}{1 - q.\alpha} - \frac{q.\alpha}{1 - q.\alpha} \cdot SimSUB(o'.cell, q) \quad (7)$$

Therefore, for the current status o', a query can be pruned by comparing the keyword similarity to the threshold instead of calculating the whole spatial keyword score.

Group threshold. According to Equation (7), the calculation of a threshold for a single query can be divided into two parts: $\frac{kScore(q)}{1-q.\alpha}$ and $\frac{q.\alpha}{1-q.\alpha} \cdot SimSUB(o'.cell, q)$. For simplicity, we denote $\frac{kScore(q)}{1-q.\alpha}$ as $\tilde{K}$ and $\frac{q.\alpha}{1-q.\alpha} \cdot SimSUB(o'.cell, q)$ as $\tilde{D}$ respectively. Then the lower bound is derived as the group threshold as:

$$T_\theta(o'.cell, g) = Min_{q \in g}\{\tilde{K}\} - Max_{q \in g}\{\tilde{D}\} \quad (8)$$

$\tilde{D}$-based partition. To generate a tighter group threshold, we should group queries with similar $T_\theta(o'.cell, q)$. The value of $\tilde{D}$ is irrelevant for the spatial attributes of objects. Moreover, $\tilde{D}$ is a static value because queries will not change their location and α, but $\tilde{K}$ will change as the top-k list updated. By intuition, we group the indexed queries in a cell by their $\tilde{D}$ value, such that the queries inside a group have similar $\tilde{D}$'s. We employ a quantile-based method to partition the domain of $\tilde{D}$ to ensure the tight grouping. Figure 4 also gives the image of a query group in a cell.

Maximum keyword similarity. On the other hand, the maximum keyword similarity between o' and g, which is denoted as $maxT(g, o')$, can be estimated as:

$$maxT(g, o') = SimT(g.maxwts, o'.\psi) \quad (9)$$

In Equation (9) $g.maxwts$ denotes the maximum weights of all keywords contained by the queries in g. (The similar mechanism exists for $maxwts$ of all objects in a cell.) As shown in Figure 4, the information of $maxwts$ is kept for each query group in our index.

Finally, we can conduct Theorem 1 to prune a group queries safely. Algorithm 1 concludes our processing of AQF. The complexity of condition checking in Theorem 1 is $O(1)$, since we can precompute the values of $\tilde{K}$ and $\tilde{D}$.

Theorem 1: For a group of queries g, and a dynamic object o', all queries in g will not be present in InQ of object o' if $maxT(g, o') < T_\theta(o'.cell, g)$.

VI. Top-k refiller

When a dynamic object o changes to o', after obtaining the affected queries, i.e., OutQ and InQ, from the *Affected Queries Finder*, the following step is to update their top-k lists. Update the top-k's for queries of InQ is low-cost since only the top-k list is considered with the o'. In other words, just needs to insert o' into the previous top-k. However, for the queries of

Algorithm 1 AQF (Affected Queries Finder)

Input: o'
1: Check object table with o' and get OutQ.
2: InQ= $\emptyset$
3: **for** each group $g \in G$ of $o'.cell$ **do**
4: **if** $maxT(g, o') > g.T_\theta$ **then**
5: **for** each query $q \in g$ **do**
6: **if** $SimST(o', q) > kScore(q)$ **then**
7: InQ= InQ$\cup\{q\}$
8: **return** OutQ, InQ

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.

In this section, we first describe a straightforward solution (GCL) that maintains a sorted cell list (CL) to retrieve the latest top-k result efficiently. We also propose a novel solution (GPCL) with a partial cell list (PCL). Different from CL, GPCL can avoid the redundant maintenance and computations by maintaining only a few cells and refilling the candidate objects to previous top-k list.

A. GCL *method with the sorted cell list*

The related research such as $kmax$ [9] and SKYPE [6] focus 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. Comparing to maintain objects, to maintain cells have advantages that: (a) the number of cells is much smaller than the number of objects; (b) cells have fixed spatial similarity bounds and infrequent changing keyword similarity bounds (only changed when an object affect the $maxwts$ or $minwts$ of a specific cell).

We use *q.CL* to denote the sorted cell list w.r.t the query q. Figure 5 gives an example of CL. All of the 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:

$$maxscore(c, q) = q.\alpha \cdot SimSUB(c, q.\rho) + (1 - q.\alpha) \cdot SimT(c.maxwts, q.\psi) \quad (10)$$

The $maxscore(c, q)$ is the upper bound, because $SimSUB(c, q.\rho)$ dominates the spatial similarity and $SimT(c.maxwts, q.\psi)$ dominates the inner product value of the keyword similarity.

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]. Algorithm 2 shows the top-k reevaluation method with CL (GCL). The complexity of searching top-k is $log(|O|)$, and the maintenance cost of CL is $o(log(n^2))$. Note that CL should be updated w.r.t the cells covering the dynamic object (Line 1) before updating the top-k lists (Line 2 - 5).

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

Algorithm 2 GCL

Input: o, o', $OutQ$, InQ
1: Update $q.CL$ w.r.t $o.cell$, $o'.cell$
2: **for** each $q \in \{InQ\}$ **do**
3: $top\text{-}k(q)$.insert(o')
4: **for** each $q \in \{OutQ - InQ\}$ **do**
5: $top\text{-}k(q)$ = Branch-and-bound top-k searching with $q.CL$

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

Including the above GCL method, previous works always focus on the top-k reevaluations [6], [10], [14]. However, these techniques contain redundant computations because they ignore the existing order and recreate a new top-k list again and again. Actually, when an object changed, only one ranking position changes, and the relative order of the other objects remains. For example, suppose that there are five objects, if the 3rd becomes the 5th, then the ranking relationship of the current 1st, 2nd, 3rd, and 4th objects do not change. Therefore, if an object changes out of the top-k, the correct top-k list can be obtained by only refilling the candidate $(k + 1)$-th object. Another limitation of GCL is the maintenance, since 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.

We use both $maxscore$ (Equation 10) and $minscore$ to index the cells in PCL. The $minscore(c, q)$ denotes the minimum spatial keyword score between any objects in cell c and a query q. $minscore$ is formed with the minimum spatial similarity ($SimSLB$) and the minimum keyword similarity ($SimTLB$).

$$minscore(c, q) = \alpha \cdot SimSLB(c, q.\rho) + (1 - \alpha) \cdot SimTLB(c.minwts, q.\rho) \quad (11)$$

$SimSLB$ is calculated similar to Equation (6), where $maxD(.)$ represents the maximum Euclidean distance.

$$SimSLB(c, q) = 1 - \frac{maxD(c, q.\rho)}{MaxDist} \quad (12)$$

and $SimTLB$ is:

$$SimTLB(c.minwts, q.\rho) = MIN_{w \in q.\psi \cap c.minwts} wt(q.w) \cdot wt(c.w) \quad (13)$$

Eventually, the $minscore$ is defined by summing $SimSLB$ and $SimTLB$:

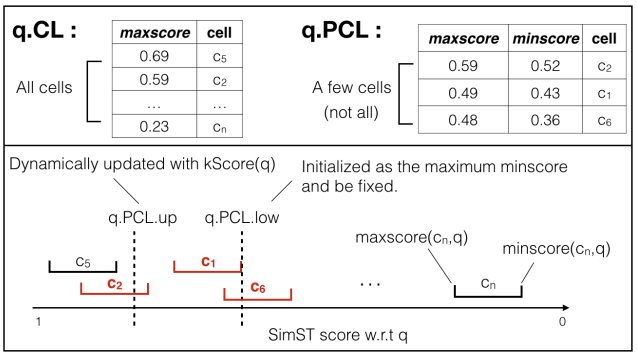


Fig. 5: Examples of CL and PCL.

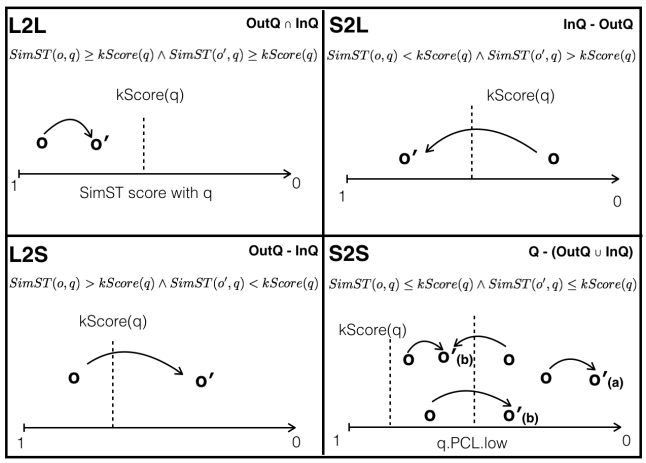


Fig. 6: Four cases of an dynamic object and a query.

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, $q.PCL.up$ can be updated as the 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 $maxscore(c_j,q) < kScore(q)$.

Theorem 2: After initialization, $q.PCL$ contains the (k+1)-th object w.r.t q.

Proof: By contradiction. Assume that the (k+1)-th object o_{k+1} is not in the cells of PCL, we can know that $SimST(o_{k+1},q) < q.PCL.low$. According to the features of PCL, at least one object o_i whose score is greater than $q.PCL.low$ must exist. Therefore $SimST(o_i,q) > q.PCL.low > SimST(o_{k+1},q)$, and o_i should be the (k+1)-th object, leading to the contradiction. ■

Example 2: Figure 5 gives an example of PCL. In this Figure, the horizontal axis shows the score distribution of all cells (c_1, c_2, ..., c_n) w.r.t a specific q. A segment represents the score range of a cell. The left side is the $maxscore$ while the right side is the $minscore$. CL is a cell list sorted by $maxscore$. Therefore. the order of $q.CL$ in this example is: $\{c_5, c_2, c_1, c_6, ..., c_n\}$. For PCL, $q.PCL.up$ is initialized as the current $kScore(q)$. On the right side of $kScore(q)$, cell c_1 has the maximum $minscore$. Thus, $q.PCL.low$ will be initialized as $minscore(c_1,q)$. Then $q.PCL$ contains the $\{c_2, c_1, c_6\}$ overlapping the range $(q.PCL.up, q.PCL.low)$ The size of PCL is much smaller than that of CL.

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

2) **PCL Maintenance:** We divide the situations between a dynamic object and a query into following four cases. Figure 6 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 PCL although the top-k list is not changed, we should maintain PCL carefully to ensure it always contains the (k+1)-th object to refill. We propose a sophisticated maintenance process that corresponds to the above four cases. Table III summarizes the operations of these cases.

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, PCL still contains the candidate object. We just *check* the changing cells $o.cell$ and $o'.cell$ with PCL. Algorithm 3 describes the procedure of *check*. In short, each input cell, according to their new $maxscore$ and $minscore$, will be inserted into (or delete from) 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 $(k+1)$ one, so we should add the $o_k.cell$ into PCL to ensure the candidate object. Since $o.cell$ and $o'.cell$ also require a *check*, we need *check* $\{o.cell, o'.cell, o_k.cell\}$. In conclusion, we employ a *check* operation to maintain PCL in the cases of L2L and S2L.

Lemma 1: Given a dynamic object and a query. If they are in the cases L2L and S2L, we need to maintain PCL with the operation of check.

Algorithm 3 q.PCL.check

Input: $Cells$
1: **for** each cell $c \in Cells$ **do**
2: **if** $c \in PCL$ **then**
3: delete c from q.PCL
4: **if** $(maxscore(c,q) > q.PCL.low)$ and $(minscore(c,q) < kScore(q))$ **then**
5: $q.PCL.insert(c)$

L2S. In the situation of L2S, we must *search* the top-1 candidate object o_{cand} from PCL. Then the scores $SimST(o',q)$ with $SimST(o_{cand},q)$ are compared to determine whether to refill o_{cand} or not. Then, we should *trim* the PCL with a

Algorithm 4 $q.PCL.trim$

Input: $kScore(q)$, $q.PCL.Low$

1: $PCL_{new} = \emptyset$
2: **for** each $c \in q.PCL$ **do**
3: **if** $(maxscore(c,q) > q.PCL.low)$ and $(minscore(c,q) < kScore(q))$ **then**
4: $PCL_{new}.insert(c)$
5: $q.PCL = PCL_{new}$

new score range $(kScore(q)', q.PCL.low)$ where $kScore(q)'$ denotes the updated k-th score. Algorithm 4 gives the details of the operation *trim*, which filters the cells that are not within the input score range. Sometimes PCL may become empty (no object) after the *trim*, then PCL must be recreated.

Lemma 2: Given a dynamic object and a query of case L2S, we need to maintain PCL with the operations of check and trim. Also, if PCL has empty candidates after trimming in case L2S, PCL must be recreated.

S2S. S2S is divided into two sub-cases: S2S.a and S2S.b. In S2S.a, both o and o' are on the outside of $q.PCL.low$, so PCL does not need to maintain. 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 5 GPCL

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

1: // **L2L**
2: **for** each $q \in OutQ \cap InQ$ **do**
3: Update $top\text{-}k(q)$ with o'.
4: $q.PCL.check(\{o.cell \cup o'.cell\})$
5: // **S2L**
6: **for** each $q \in InQ - OutQ$ **do**
7: Update $top\text{-}k(q)$ with o'.
8: $q.PCL.check(\{o.cell \cup o'.cell \cup k.cell\})$
9: // **L2S**
10: **for** each $q \in OutQ - InQ$ **do**
11: o_{cand} = Retrieve Top-1 from $q.PCL$
12: **if** $SimST(o',q) > SimST(o_{cand},q)$ **then**
13: Update $top\text{-}k(q)$ with o'.
14: **else**
15: Update $top\text{-}k(q)$ with o_{cand}.
16: $q.PCL.check(\{o.cell \cup o'.cell\})$
17: $q.PCL.trim(kScore(q)', q.PCL.low)$
18: **if** $q.PCL$ is empty **then**
19: $q.PCL.recreate$
20: // **S2S**
21: **for** each $q_i \in Q - OutQ \cup InQ$ **do**
22: **if** $SimST(o,q) < q.PCL.low$ and $SimST(o',q) < q.PCL.low$ **then**
23: **continue**
24: **else**
25: $q.PCL.check(\{o.cell \cup o'.cell\})$
26: **if** $q.PCL$ is empty **then**
27: $q.PCL.recreate$

Lemma 3: Given a dynamic object and a query, if they are in case S2S.a, PCL does not need to be maintained. On the other hand in S2S.b, PCL must be maintained with the *check* operation. If PCL has no candidates, PCL must be recreated.

	Operation on PCL			
Case	Search	Check	Trim	Re-create
L2L	-	✓	-	-
S2L	-	✓	-	-
L2S	✓	✓	✓	✓(if empty)
S2S.a	-	-	-	-
S2S.b	-	✓	-	✓(if empty)

TABLE III: Summary of the operations for PCL maintenance.

Algorithm 5 gives the pseudo-code of proposed candidate refilling method GPCL method with the sophisticated maintenance based on Lemmas 1, 2 and 3.

VII. Theoretical analysis

Compared to CL, PCL is advantageous with regard to a (k+1)-th maintenance. On the other hand, if PCL is empty, it must be recreated from all cells, which is an expensive operation ($O(n^2 logn^2)$). Therefore, there is a trade-off between the number of cells and recreation. Fewer cells in PCL realize an efficient search and maintenance, but lead to a higher probability that an expensive recreation will be triggered. Obviously, the number of cells in PCL depends on the number of cells n^2 in the grid. In this section, we conduct a theoretical analysis to build a cost model that balances this trade-off by estimating the best value of n^2.

The total expecting cost is the sum of the expected costs of each operation.

$$E[total] = E[check] + E[trim] + E[search] + E[recreate] \tag{14}$$

The expected cost of an operation can be calculated by multiplying the probability to the number of the similarity computing. According to Table III, the expected costs of all operations can be calculated by the equations below. Here, $P(X)$ means the probability that event X happens, PCL_c represents the cells number in PCL, and PCL_o means the objects number in PCL.

$$E[check] = (1 - P(S2S.a)) \cdot log(PCL_c) \tag{15}$$

$$E[trim] = P(L2S) \cdot PCL_c \tag{16}$$

$$E[search] = P(L2S) \cdot log(PCL_o) \tag{17}$$

$$E[recreate] = P(PCL.empty) \cdot n^2 \cdot logn^2 \tag{18}$$

To estimate $E[total]$, we need to find the probabilities of $P(S2S.a)$, $P(L2S)$ and $P(PCL.empty)$. We implement a theoretical analysis based on the following assumption. The queries and objects follow a uniform distribution in [0,1) space. With n^2 cells, the length of a cell is $l = \frac{1}{n}$, and the average number of objects in a cell is $\frac{|O|}{n^2}$, where $|O|$ is the size of the object set.

For an object, we assume that a spatial (maximum) step in space is δ_s. To evaluate the similarities on spatial and keywords, we convert the value on one to the other. First, we estimate δ_t, which is calculated by computing the change on the keyword similarity and convert it to space. We use w_q

and w_o to represent the average number of keywords in an object and a query, respectively. The change of the keyword similarity is

$$SimT(o'.\psi, q.\psi) - SimT(o.\psi, q.\psi) = \sum_{w \in o.\psi \cap q.\psi} wt(q.w) \cdot (wt(o'.w) - wt(o.w)) \approx \frac{1}{w_q^2 \cdot w_o^2} \quad (19)$$

By recalling that the distance on [0,1) space ranges is from 0 to $\sqrt{2}$, while the similarity on the keywords is the cosine value, we convert the change of keyword similarity to that on the space by multiplying the ratio between the two ranges as

$$\delta_t = \frac{\sqrt{2}}{w_q^2 \cdot w_o^2}. \quad (20)$$

Then we estimate the probabilities. Recall that L2S is the case where an object belongs to top-k and changes out of top-k. As shown in Figure 7a, $P(L2S)$ can be estimated as the ratio of the areas, which is expressed as:

$$P(L2S) = \frac{S_{\delta_1}}{S_k + S_{\delta_1}} \cdot S_k = \pi {r_k}^2 - \frac{\pi r_k^4}{(r_k + \delta_s + \delta_t)^2} \quad (21)$$

Since we assume that all points follow a uniform distribution, the ratio of the area between the top-k circle and the whole space represents the ratio of the objects numbers. Hence, the radius r_k in Equation (21) can be calculated as:

$$\frac{\pi {r_k}^2}{1} = \frac{k}{|O|} \implies r_k = \sqrt{\frac{k}{\pi |O|}} \quad (22)$$

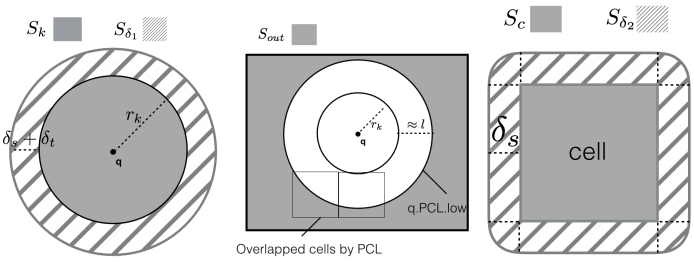
According to Figure 7b, P(S2S.a) is computed based on the areas in a similar way.

$$P(S2S.a) = {S_{out}}^2 = (1 - \pi(r_k + l)^2)^2 \quad (23)$$

The number of cells in PCL, PCL_c, is estimated by the number of cells that overlap the circle of q.PCL.low. As shown in Figure 7b, the number of overlapped cells is approximately the value of the perimeter divided by side length of a cell (l). That is, $PCL_c = \frac{2\pi(r_k + l)}{l}$. Then we can calculate the objects number $PCL_o = PCL_c \cdot \frac{|O|}{n^2}$. Because PCL has PCL_o objects, the probability of PCL only has one object can be simply approximated as $\frac{1}{PCL_o}$. When PCL contains only one object, PCL may become empty when this object moves out of its cell. Note that moving out of a cell depends only on δ_s. Figure 7c shows the areas of this situation. In conclusion, $P(PCL.empty)$ is calculated as:

$$P(PCL.empty) = \frac{1}{PCL_o} \cdot \frac{S_{\delta_2}}{S_{\delta_2} + S_c} = \frac{1}{PCL_o} \cdot \frac{4l\delta_s + \pi\delta_s^2}{4l\delta_s + \pi\delta_S^2 + l^2} \quad (24)$$

Eventually, with the parameters $|O|$, k, δ_s, w_q and w_o, the only variable in Equation (14) is n. We used the gradient descent to figure out the n by computing the extreme value to minimize $E[total]$.



(a) L2S. (b) S2S.a. (c) Out of a cell.

Fig. 7: Areas.

VIII. Experiments

We conducted sufficient experiments to verify the efficiency of our proposal. All algorithms were implemented in C++ on a Mac with a 2.2GHz Intel Core i7 CPU and 32GB memory. All indices, buffers, and algorithms were run on in-memory.

A. Setting

Dataset. We used two real datasets and one synthetic dataset. Table IV shows the statistics of the datasets. **YELP** is an open source dataset provided by YELP web service[3]. It contains 1,100,000 of reviews on 156,000 businesses by 220,000 users. Each user has at least four reviews of different businesses.

We set the businesses with their locations and descriptions as the queries. For the keyword attributes of the queries, we randomly selected 1 to 5 keywords from the description. We set the first review of each user as the initial states of an object. Unfortunately, the reviews data did not contain users' location information. Thus, we intuitively set the initial location of the objects as the business location in the review. For the future states of the dynamic objects, we set a simple random walk that randomly added or subtracted 0.05 to the previous coordinates. Then we used the contents of other reviews from the same user as the changes in the keyword attribute.

TWITTER is the dataset with 4,200,000 geo-tag tweets from the United States[4], which are extracted via Twitter REST API from Twitter[5] and the

In TWITTER, there are 1.2M unique users. Each user has at least three geo-tag tweets.

First, we randomly selected 100K to 500K users and used their first tweets as the queries in the experiment. For each query, we set the spatial attribute as their geo-tag. Then select 1 to 5 words from the tweets as the keyword attributes. On the other hand, we randomly selected 200K to 1M from the remaining unique users and initialized the objects with their first geo-tag tweets. The rest tweets from selected users were treated as the continuous states of moving objects.

SYN is a synthetic data containing 12M spatial keyword tuples. For the spatial attributes, we used the data of moving

[3]https://www.yelp.com/dataset
[4]https://datorium.gesis.org/xmlui/handle/10.7802/1166
[5]https://twitter.com/

Datasets	YELP	TWITTER	SYN
Data size	1.1M	4.2M	12M
Default # of selected objects	220K	500K	1M
Default # of selected queries	156K	250K	1M
# of keywords	759K	3.5M	819K
# of average keywords	5.4	4.5	3

TABLE IV: Datasets statistics

points, [6] which generated by BerlinMOD benchmark [23]. For the keyword attributes, we randomly selected 1 to 5 keywords from the TWITTER dataset and assigned them to each point. Objects and Queries were selected from the SYN tuples.

Algorithms. For the *affected queries finder* module, we compared the following methods:
• **CIQ.** The block-based inverted file structure in [5].
• **IGPT.** The group pruning techniques in [6].
• **AQF.** The proposed method in this paper with a influence circle and a group pruning technique.

For the *top-k refiller* module, we compared:
• **kmax**. The method with $kmax$ buffer in [9]. For the size of the buffer, we tuned the value of $kmax$ from k to $5k$. The experimental results lead us to set $3k$ as the default value of $kmax$. The tuning result is similar to related work [6].
• **GCL.** The proposed method with sorted cell list in Algorithm 2.
• **GPCL.** The proposed method with partial sorted cell list in Algorithm 5.

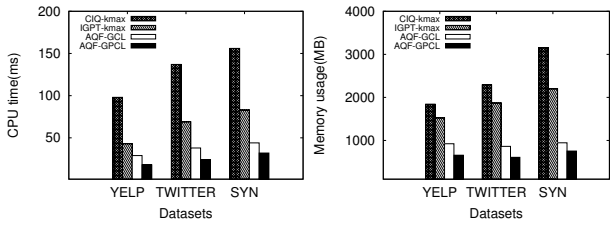
We randomly selected over 10,000 status of dynamic objects and reported the average processing time and memory usage among different indices of above algorithms. The default result size k was 100 and the smoothing parameter α was a random value in (0,1).

B. Experimental Results

Comparison of overall processing. Figure 8 shows the comparison results for the overall processing with the default size of objects and queries shown in Table IV. Specifically, we compared the two proposed methods, AQF-GCL and AQF-GPCL to two related works. Note that AQF-GCL means that we imported the AQF algorithm as the *affected queries finder* module, and utilize GCL algorithm as the *top-k refiller* module. For the related works IGPT and CIQ, we adjusted their techniques with the $kmax$ method so that they can deal with our problem of dynamic objects. Both of our proposed methods, AQF-GCL and AQF-GPCL, have better performances than the others with respect to the processing time and memory cost. The following observations in which the two modules are independently tested reveal the reason for the efficacy.

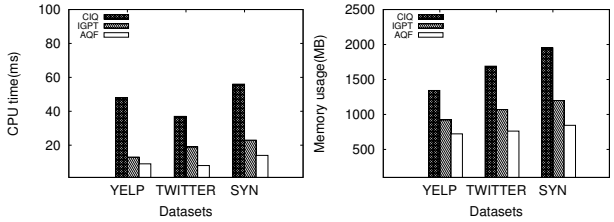
Comparison results of finding affected queries. Figure 9 shows the comparison results of finding affected queries. Out method AQF is at least 1.5 times faster than the other methods. The reason is because IGPT and CIQ required overhead costs on the traversal of the quad-tree index from the root node. In contrast, our grid-based index can index only a few

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



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

Fig. 8: Overall processing.



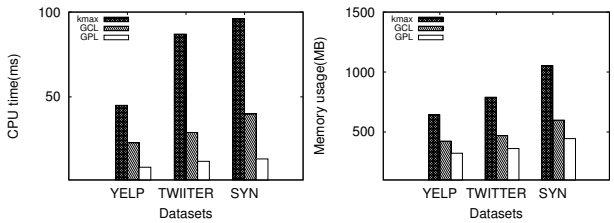
(a) Average processing time. (b) Memory usage.

Fig. 9: *Affected queries finder* module.

candidate queries and retrieve them directly (o(1) complexity). Moreover, the group pruning technique in AQF also boosts the performance by filtering unnecessary queries.

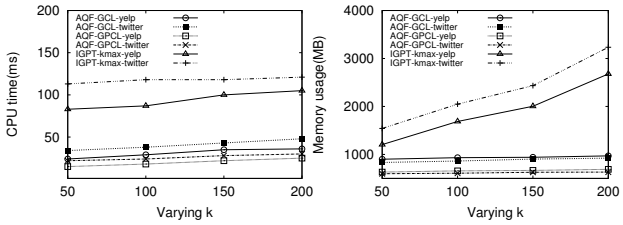
The memory usage of our grid-index for queries is the smallest one (Figure 9b). We only index each query once, while IGPT indexes queries several times with keywords in the inverted files, and CIQ indexes queries in multiple nodes of the quad-tree.

Comparison results for top-k reevaluation. To test our *top-k refiller*, we compare the proposed GCL and GPCL methods with the $kmax$ method. The results include two parts: the processing time of top-k reevaluation and the maintenance time. According to Figure 10, our proposed methods are at least twice as fast as the $kmax$. The GPCL method is better than GCL since it uses the candidate refilling strategy rather than reevaluating the whole top-k. Moreover, PCL also balances the trade-off that only maintains a few cells. Therefore, GPCL has a least memory usage.



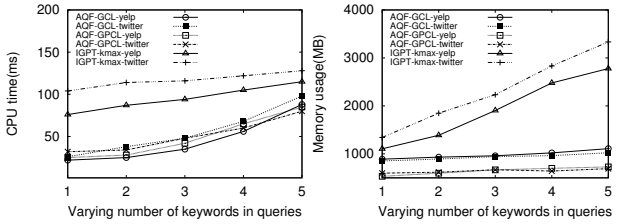
(a) Average processing time. (b) Memory usage.

Fig. 10: *Top-k refiller* module.



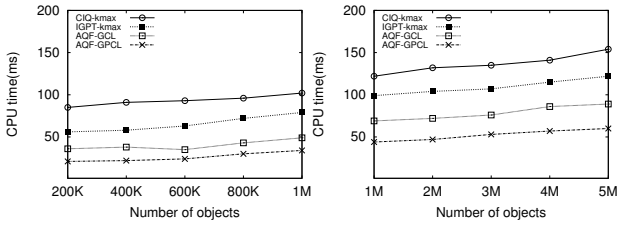
(a) Average processing time. (b) Memory usage.

Fig. 11: Varying k.



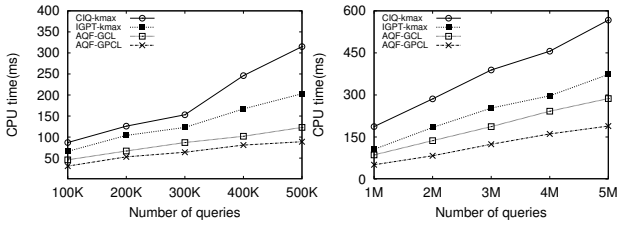
(a) Average processing time. (b) Memory usage.

Fig. 12: Varying number of keywords in queries.



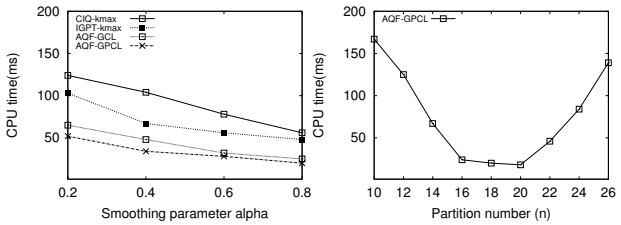
(a) TWITTER data. (b) SYN data.

Fig. 13: Varying number of objects.



(a) TWITTER data. (b) SYN data.

Fig. 14: Varying number of queries.



(a) TWITTER data. (b) TWITTER data.

Fig. 15: Varying n and α.

Effect on searching results size k. According to the Figure 11a, AQF-GPCL is the only one algorithm not influenced by k since the mechanism of GPCL refills the candidate (k+1)-th object rather than reevaluates the whole top-k. From the memory usage of indices in Figure 11b, a large k leads to a bigger index for $kmax$-based methods. However, our proposed methods are unaffected by k since we index the identity of the cells. As we introduced previously, PCL can be seen as a small subset of CL. Hence, AQF-GPCL has a smaller memory cost than AQF-GCL.

Effect on the number of query keywords. Figure 12 shows the processing performance among different methods with a varying number of keywords in queries. Our methods are better than the other methods in all situations for a given number of keywords. The processing time of our proposed methods is close to other methods as the number of keywords increases because many keywords will loose the bound of spatial keyword similarity of a cell in Equations (10) and (11). Figure 12b shows the memory usage of varying number of keywords in queries. Our methods keep their superior and AQF-GPCL costs the smallest memory among all indices (buffers). The inverted-file leads IGPT-based and CIQ-based methods cost more space to index data with more keywords.

Effect on number of objects and queries. Figure 13 shows the effects of varying the cardinality of objects with TWITTER data and SYN data. Since all algorithms keep objects with spatial indexes and implement a buffer to maintain the candidate objects, they are not affected too much by the large size of objects. However, as the Figure 14 shows, a large size of queries affects the efficacy of all algorithms because more queries are affected by a dynamic object, which subsequently triggers more processing to update the results.

Effect on smoothing parameter α. Figure 15a shows the results of varying α. All algorithms have better performance in higher α since the pruning power is mainly on the spatial attribute.

Effect on number of cells n^2 in the grid-based index. To test the precision of the cost model in Section VII, we used the TWITTER data and varied n^2 to observe the processing performance with the different number of cells. The optimal theoretical results of n by minimizing Equation (14) is 15.44. In the experiment in Figure 15b, the best result of n is 20, while results of 16 and 18 are close to the optimal value. We conclude that our cost model can define a good n for the grid-based index, realizing an optimal performance.

IX. Conclusion

Numerous web services such as location-based advertisements and target tracking may benefit from the techniques that searching dynamic special keyword objects continuously. In this paper, we investigate a novel problem that searching dynamic spatial keyword objects continuously and propose a solution system. We employ a grid-based index to handle both dynamic objects and queries. To efficiently detect the affected queries by an object efficiently, we propose a group pruning strategy in our *affected queries finder* module. To maintain the top-k results with a quick-response and low-cost, we propose a sophisticated buffer called the partial cell list (PCL) to efficiently refilling 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.

As a future work, we plan to propose a distributed process based solution to enhance the performance of our system.

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, C. S. Jensen, and G. Cong, "Efficient continuously moving top-k spatial keyword query processing," in *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, 2011, pp. 541–552. [Online]. Available: https://doi.org/10.1109/ICDE.2011.5767861

[2] 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. [Online]. Available: http://doi.acm.org/10.1145/2445583.2445590

[3] W. Huang, G. Li, K. Tan, and J. Feng, "Efficient safe-region construction for moving top-k spatial keyword queries," in *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, 2012, pp. 932–941. [Online]. Available: http://doi.acm.org/10.1145/2396761.2396879

[4] 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. [Online]. Available: https://doi.org/10.1109/ICDE.2016.7498297

[5] 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*, 2015, pp. 255–266. [Online]. Available: https://doi.org/10.1109/ICDE.2015.7113289

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

[7] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *PVLDB*, vol. 2, no. 1, pp. 337–348, 2009. [Online]. Available: http://www.vldb.org/pvldb/2/vldb09-677.pdf

[8] K. V. R. Kanth, S. Ravada, and D. Abugov, "Quadtree and r-tree indexes in oracle spatial: a comparison using GIS data," in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6, 2002*, 2002, pp. 546–557. [Online]. Available: http://doi.acm.org/10.1145/564691.564755

[9] 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, March 5-8, 2003, Bangalore, India*, 2003, pp. 189–200. [Online]. Available: https://doi.org/10.1109/ICDE.2003.1260792

[10] 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. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142544

[11] X. Yu, K. Q. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, 2005, pp. 631–642. [Online]. Available: https://doi.org/10.1109/ICDE.2005.92

[12] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias, "Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, 2005, pp. 634–645. [Online]. Available: http://doi.acm.org/10.1145/1066157.1066230

[13] X. Xiong, M. F. Mokbel, and W. G. Aref, "SEA-CNN: scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases," in *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, 2005, pp. 643–654. [Online]. Available: https://doi.org/10.1109/ICDE.2005.128

[14] H. Hu, J. Xu, and D. L. Lee, "A generic framework for monitoring continuous spatial queries over moving objects," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, 2005, pp. 479–490. [Online]. Available: http://doi.acm.org/10.1145/1066157.1066212

[15] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-k spatial keyword queries," in *Advances in Spatial and Temporal Databases - 12th International Symposium, SSTD 2011, Minneapolis, MN, USA, August 24-26, 2011, Proceedings*, 2011, pp. 205–222.

[16] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *PVLDB*, vol. 6, no. 3, pp. 217–228, 2013. [Online]. Available: http://www.vldb.org/pvldb/vol6/p217-chen.pdf

[17] G. Cong and C. S. Jensen, "Querying geo-textual data: Spatial keyword queries and beyond," in *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, 2016, pp. 2207–2212. [Online]. Available: http://doi.acm.org/10.1145/2882903.2912572

[18] A. R. Mahmood and W. G. Aref, "Query processing techniques for big spatial-keyword data," in *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, 2017, pp. 1777–1782. [Online]. Available: http://doi.acm.org/10.1145/3035918.3054773

[19] L. Guo, J. Shao, H. H. Aung, and K. Tan, "Efficient continuous top-k spatial keyword queries on road networks," *GeoInformatica*, vol. 19, no. 1, pp. 29–60, 2015. [Online]. Available: https://doi.org/10.1007/s10707-014-0204-8

[20] G. Li, Y. Wang, T. Wang, and J. Feng, "Location-aware publish/subscribe," in *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, 2013, pp. 802–810. [Online]. Available: http://doi.acm.org/10.1145/2487575.2487617

[21] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang, "Ap-tree: efficiently support location-aware publish/subscribe," *VLDB J.*, vol. 24, no. 6, pp. 823–848, 2015. [Online]. Available: https://doi.org/10.1007/s00778-015-0403-4

[22] L. Chen, G. Cong, and X. Cao, "An efficient query indexing mechanism for filtering geo-textual data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, 2013, pp. 749–760. [Online]. Available: http://doi.acm.org/10.1145/2463676.2465328

[23] C. Düntgen, T. Behr, and R. H. Güting, "Berlinmod: a benchmark for moving object databases," *VLDB J.*, vol. 18, no. 6, pp. 1335–1368, 2009. [Online]. Available: https://doi.org/10.1007/s00778-009-0142-5