A Unified Processing Paradigm for Interactive Location-based Web Search

Sheng Wang, Zhifeng Bao, Shixun Huang
RMIT University
firstname.secondname@rmit.edu.au

Rui Zhang
The University of Melbourne
rui.zhang@unimelb.edu.au

ABSTRACT

This paper studies the location-based web search and aims to build a unified processing paradigm for two purposes: (1) efficiently support each of the various types of location-based queries (kNN query, top-k spatial-textual query, etc.) on two major forms of geo-tagged data, i.e., spatial point data such as geo-tagged web documents, and spatial trajectory data such as a sequence of geo-tagged travel blogs by a user; (2) support interactive search to provide quick response for a query session, within which a user usually keeps refining her query by either issuing different query types or specifying different constraints (e.g., adding a keyword and/or location, changing the choice of k, etc.) until she finds the desired results. To achieve this goal, we first propose a general Top-k query called Monotone Aggregate Spatial Keyword query-**MASK**, which is able to cover most types of location-based web search. Next, we develop a unified indexing (called Textual-Grid-Point Inverted Index) and query processing paradigm (called ETAIL Algorithm) to answer a single **MASK** query efficiently. Furthermore, we extend ETAIL to provide interactive search for multiple queries within one query session, by exploiting the commonality of textual and/or spatial dimension among queries. Last, extensive experiments on four real datasets verify the robustness and efficiency of our approach.

ACM Reference format:
Sheng Wang, Zhifeng Bao, Shixun Huang and Rui Zhang. 2018. A Unified Processing Paradigm for Interactive Location-based Web Search. In *Proceedings of WSDM'18, February 5–9, 2018, Marina Del Rey, CA, USA, ,* 9 pages.
DOI: http://dx.doi.org/10.1145/3159652.3159667

1 INTRODUCTION

The widespread use of mobile devices with GPS enables a parade of geo-tagged web data being continually generated. Such data usually behaves in two forms: (1) a single point such as a geo-tagged document in Trip-advisor[1], (2) a trajectory which is a set or sequence of geo-tagged documents, such as a user's check-in records from Twitter[2] [42], a sequence of user's travel blogs[3], etc.

[1]https://www.tripadvisor.com/
[2]https://www.twitter.com/
[3]https://www.travelblog.org/

WSDM'18, February 5–9, 2018, Marina Del Rey, CA, USA
 ISBN 978-1-4503-5581-0/18/02…$15.00
DOI: http://dx.doi.org/10.1145/3159652.3159667

Numerous types of *location-based web search* queries [4, 8–12, 22, 23, 30, 35–37, 40, 41, 43, 47] have been proposed to retrieve such geo-tagged web data, to further support various location-aware services such as site selection, trip planning, etc, as shown in Example 1.1.

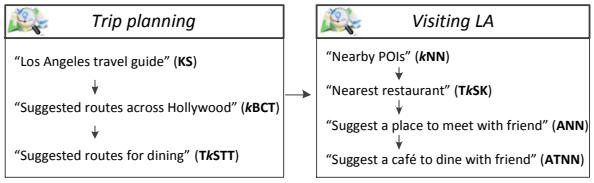

Figure 1: Exemplar queries conducted before & during a trip

Example 1.1. Grace is planning to visit *Los Angeles*, she starts her exploration by a pure keyword query (**KS**) [5] on the web to acquire initial knowledge of the famous attractions. Then she might issue a k-Best-Connected-Trajectory (k**BCT**) query [9] to find the top-k best routes (from existing travel blogs) which covers the attractions in her wish-list. Furthermore, with the Top-k Spatial-Textual Trajectory (**TkSTT**) query [30, 35, 42], she can add more keyword constraints such as preferred activities to do in the trip, to find a more personalized route to follow. After arriving in LA, Grace wants to explore the POIs around her hotel. Initially, she may do a k-Nearest Neighbor (k**NN**) query [19] search as she does not figure out which keywords to use for searching. After getting the results, she may use some keywords to add more constraints to further filter unrelated objects. It is known as Top-k Spatial Keyword (**TkSK**) query [12, 40]. Besides, she may want to find a place to meet her friends attending WSDM, which is known as Aggregate Nearest Neighbour (**ANN**) query [24] and Aggregate Textual Nearest Neighbour (**ATNN**) query [37].

It is desired to support the above queries (in Figure 1) related to location-based web data in one map-based search interface, say Google Maps (although it only supports keyword query so far). In a query session, a user may issue several queries of the same type (with different settings on textual or spatial dimension), or even different types of queries, along her exploration of the unknown data, until finding the desired objects for different purposes. Therefore, it is critical to provide quick response to various queries (probably issued in one query session), to achieve an interactive search experience [21, 43]. A straightforward approach is to adopt the above standalone work to handle each type of query; however, it is expensive in both storage and computation cost, thus failing to provide interactive search. Furthermore, an extensive literature study shows that, *no existing work can support any of the above queries over both point data and trajectory data simultaneously.*

Therefore, the first goal is to build a unified index and query processing paradigm to answer various types of location-based search queries on both spatial point data and spatial trajectory data. The second goal is to support interactive search to provide quick

response for a query session consisting of different types or settings of query. We made the following contributions to achieve them:

- By studying those most common top-k queries over geo-tagged objects in academia comprehensively (Section 2), we propose the Monotone Aggregate Spatial Keyword query-**MASK** which can represent most of them (Section 3).
- We develop an efficient search framework for **MASK**, including an index called **TGP** composed by grid-index and inverted lists tightly, and an efficient search paradigm-ETAIL (Section 4).
- We highlight three common refinements of the queries over geo-tagged web data, and propose an optimized RETAIL by reusing scanned objects, to achieve an interactive search experience (Section 5).
- We conduct extensive experimental evaluations over four real datasets to verify the efficiency and generality of our approach (Section 6).

2 PRELIMINARIES & RELATED WORK

Table 1: An overview of existing top-k queries over geo-tagged data. The column for "Index" only lists the representative index structure.

Query	Input	Output	Similarity	Index
KS [5, 13, 16]	keywords	points	TF-IDF	Inverted list
kNN [3, 19, 29]	point	points	Euclidean	R-tree
TkSK [11, 22, 40, 41]	point, keywords	points	Aggregate	IR-tree
ANN [24, 38, 48]	points	points	Aggregate	R-tree
ATNN [23, 37]	points, keywords	points	Aggregate	IR-tree
kBCT [9, 25, 34]	points	trajectories	Aggregate	R-tree
TkSTT [30, 35, 42]	points, keywords	trajectories	Aggregate	Grid index, Inverted list

2.1 Single-point Search

Spatial-only Query. A spatial-only query usually behaves in two forms: (1) a *k-nearest neighbours* (k**NN**) query, where given a spatial point p representing user's current location and a k, finding the kNN of p in the data repository; (2) a *range query* which finds all objects falling within a region (which could be rectangle or circle). For example, a kNN query can be "*find the 3 nearest ATMs around my hotel*", and a range query can be finding all the ATMs in city.
General Methodology: R-tree [19] is one of the most popular index structures for efficient spatial searching. By judging whether the minimum distance between the query and the node of index is bigger than the distance to the current k-th result, or whether the node intersects with the query range, a batch of objects can be skipped directly.

Keyword-only Query. The keyword-only query is also an important entry to explore geo-tagged data. For example, people might want to search nearby restaurants by keywords "*tacos restaurant*" (in Google Maps) when they go to *Los Angeles*. It usually behaves in two forms: (1) a textual boolean query (*conjunctive search*), where the objects that contain all query keywords are returned as results. However, a textual boolean query is too strict when a query has multiple keywords. (2) a Top-k Keyword query (*disjunctive search*) which returns the top-k objects ranked by the similarities of objects w.r.t. the query. Here, we focus on the top-k keyword query.

General Methodology: Term-at-a-time (TAAT) and Document-at-a-time (DAAT) are two main types of search paradigm [5] in the area of information retrieval. The inverted list on each term can be used to filter those objects that do not contain any keyword of the query. In term of the similarity metric, TF-IDF [27] and Language Model [31] are two widely adopted ones.

Spatial-keyword Query. The spatial-keyword query evolves from the spatial-only query by having a further constraint on textual dimension, and can be further divided into three general groups [7]: 1) *Boolean Range Query* [20], e.g., finding all the Mexican restaurants in the CBD; 2) *Boolean kNN Query* [14], e.g., finding 3 nearest Mexican restaurants around my hotel; 3) *Top-k Spatial-keyword Query* [12, 40], e.g., finding 3 restaurants which could serve lobster, pizza or steak and are also close to my hotel.
General Methodology: Most work [7, 40] proposed to use the inverted lists on top of an existing tree-structured index (say R-tree) to deal with spatial-keyword search. The inverted list can be as simple as indicating the existence of a particular keyword to serve the boolean part of the query, or as complex as incorporating the weight of the keywords to serve the similarity computation for Top-k spatial-keyword query.

2.2 Multiple-point Search

Search over Points. Given a set of points, Papadias et al. [24] searched k Nearest Neighbour points in term of the aggregate distance of each neighbor to query points, and it is called *Aggregate Nearest Neighbour* (ANN) search. Li et al. [23], Yao et al. [37] proposed *Aggregate Textual Nearest Neighbour* (ATNN) search based on ANN by adding keywords in query. The similarity is an aggregation of the spatial proximity to the query locations and textual similarity to the query keywords.

Search over Trajectories. Given a set of points, Chen et al. [9] proposed a *k Best Connected Trajectories* (kBCT) query, in order to find the trajectories that are among the k-closest ones to the query points. The distance between trajectory and query is an aggregation of the shortest distance from each point (in trajectory T) to the points in query. Subsequently, *Top-k Spatial-Textual Trajectory* search evolves from the spatial-only search by incorporating keywords into the query and its search method [30, 35, 42]. In particular, a keyword could be associated with either a designated query point like [35] for a fine-grained search, or associated with the whole trajectory for a coarse-grained search like [30, 42].

2.3 Query Refinement

Query refinement originates from web search, where users have to reformulate their queries 40% to 52% of the time in order to find their desired results [32]. It can be caused by many different reasons: there do not exist matching results in the data repository that matches users' search intentions [18, 21, 39], or there is a mismatch between users' search intentions and their formulated queries (due to spelling errors, limited domain knowledge, etc.) [45]. For example, Grace is searching "3 nearest restaurants", and gets 3 results which contain Mexican food that Grace likes a lot, so she further changes the query as "3 nearest Mexican restaurants".

Query refinement applies to not only textual search, but also spatial search. One type of query refinement is the moving object

query [36], where the query location keeps moving. A common methodology is to propose the concept of "safe zone", within which the up-to-date top-k results remain unchanged while it triggers a re-computation when it walks out of the safe zone. For spatial point data, the most recent work is to execute the refinement on both spatial dimension (in term of relaxing the search range) and textual dimension (via approximate string matching methods) [45].

2.4 Summary

To summarize, we observe that none of existing work can handle the single point search and multiple-point search over the point data and the trajectory data at the same time. Such limitation behaves in two-folds: a unified index to support most common query operators is missing; a unified query processing approach to handle various types of queries is missing. Moreover, it remains an open problem on how to support an interactive search where users may change the type of query or the parameters (like k, keywords) for a certain type of query.

3 PROBLEM DEFINITION

In this section, we formally define the *Top-k Monotone Aggregate Spatial Keyword Query*, which is a generalized form of most existing spatial-only and spatial-textual queries (over both the point data and trajectory data) as described in Section 2.

Definition 3.1. **(Point)** In a data repository D, a point $p = (\rho, \psi)$ is a pair consisting of the location ρ and a set of associated terms $\psi = (t_1, t_2, \ldots, t_i)$ describing ρ and/or users' activities at ρ.

Definition 3.2. **(Trajectory)** A trajectory T of length l is in the form of $\{p_1, p_2, \ldots, p_l\}$, where each p_i is a point.

As trajectory [30, 35] and point [7] are two main forms of geo-tagged data, in this paper we use object o uniformly to represent trajectory and point.

Definition 3.3. **(Query)** A query Q is a set of points in the form of $\{q_1, q_2, \ldots, q_m\}$. Specifically, $Q_s = \{\rho_1, \rho_2, \ldots, \rho_m\}$ is the query location set, and $Q_t = \bigcup_{i=1}^{m} \psi_i$ is the query keyword set, i.e., $m = |Q_s|$ and $n = |Q_t|$, m and n cannot be 0 at the same time.

By reviewing existing widely used similarity models on Top-k search [5, 9, 19, 22, 23, 27, 30, 35–37, 40, 41, 43], we can generalize them as the below definition:

Definition 3.4. **(Monotone Aggregate Similarity)** The similarity of an object o w.r.t. a query Q is defined in the below equation:

$$\hat{S}(Q, o) = \alpha \cdot \sum_{\rho \in Q_s} SProx(\rho, o.\rho) + (1 - \alpha) \cdot \sum_{\psi \in Q_t} TRel(\psi, o.\psi) \quad (1)$$

where $SProx(\rho, o.\rho)$ is the spatial proximity between $o.\rho$ and ρ, $TRel(\rho, o.\psi)$ is the textual relevance between ψ and $o.\psi$, and $\alpha \in [0, 1]$ is a query preference parameter that makes it possible to balance the spatial proximity and textual relevance, and it is always set as 0.5.

Spatial proximity. $SProx(\rho, o.\rho)$ is defined as the normalized minimum Euclidean distance from ρ to $o.\rho$, computed as:

$$SProx(\rho, o.\rho) = 1 - \frac{\min\limits_{p_j \in o.\rho} dist(\rho, p_j)}{dist_{max}} \quad (2)$$

where $dist(\rho, p_j)$ is the *Euclidean distance* between p_j and q, and $dist_{max}$ is the maximum distance between any two objects in D. Besides the Euclidean distance, the distance function can also be the *road network distance* [38]. In this paper, we focus on the more widely used Euclidean distance function.

Textual relevance. For $TRel(\psi, o.\psi)$, similar to [5, 9, 22, 23, 30, 35–37, 40, 41, 43], we use a scoring model such as TF-IDF which is also a monotone aggregate function.

$$TRel(\psi, o.\psi) = \sum_{t \in \psi \cap o.\psi} w(t, o.\psi) \quad (3)$$

where $w(t, o.\psi)$ is a normalized similarity weight of keyword t in $o.\psi$, and the maximum value of weight is 1. In our experiment, we compute the weight of each term in an object by their occurrence numbers over the number of all keywords in the object. Beside, our framework can be modified to fit other textual similarity models that can be answered efficiently using inverted list, such as Language Model [31] and BM25 [26], we leave these details for future work.

Taking the most recent spatial-textual trajectory search work [35] for example, it combines the textual and spatial similarity at point-to-point level. It is actually also an instance of Equation 1, because for each point in the query, only the data point o that maximizes the similarity is chosen to contribute to the overall similarity $\hat{S}(Q, o)$ in Equation 1.

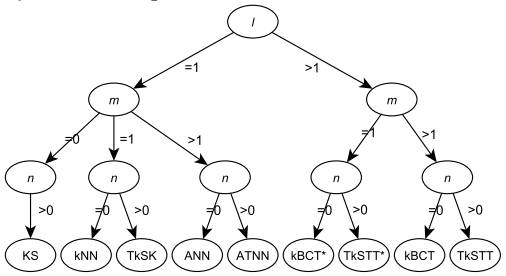

Figure 2: Different types with respect to different combination of l, m and n. "$*$" means the special case of the query.

Definition 3.5. **(Top-k Monotone Aggregate Spatial Keyword Query)** Given an object database $D = \{o_1, \ldots, o_{|D|}\}$ and query Q, a Monotone Aggregate Spatial Keyword (**MASK**) query retrieves a set $\mathcal{R} \subseteq D$ with k objects such that: $\forall r \in \mathcal{R}, \forall r' \in D - \mathcal{R}, \hat{S}(Q, r) > \hat{S}(Q, r')$.

We present how most representative spatial-only and spatial-textual queries [9, 19, 23, 24, 28, 30, 35, 37, 40] behave as a specific form of our proposed **MASK** query in Figure 2. For example, when $l = 1$, $m = 1$ and $n = 0$, it becomes a spatial-only kNN query over point data [19]; when $l > 1$, $m > 1$ and $n > 0$, it becomes a spatial-textual kNN query over trajectory data [35].

4 A UNIFIED PROCESSING PARADIGM

4.1 Preliminary

Threshold Algorithm. In order to avoid scanning every object $o \in D$ to get top-k results, the threshold algorithm (**TA**) of Fagin et al. [17] is used as a filtering method when accessing the ranked lists (sorted by the term weight of objects). The filtering method proceeds in three steps: **1)** for each query term, its ranking list is scanned sequentially so that objects with high similarity scores are scanned first; **2)** compute the similarity of scanned objects and update the top-k result heap; **3)** check whether the k-th result has a higher

score than the upper bound of the unscanned objects (Spotted area), terminate if yes, otherwise jump to Step 1. Specifically, the upper bound is the sum of score in the current rows of each inverted list. It is worth noting that the ranked list can be divided into blocks to reduce I/Os, which is known as the *block-based early termination* [15, 16].

While in principle a similar idea can be adopted to suit search purpose, using the algorithm of Fagin et al. directly does not work because the top-k list for every point in the query is not known a priori. For example, when the query in Figure 1 comes, we can get the inverted lists for "Restaurant" and "Mexican" easily, but cannot get the inverted lists for locations. Hence, how to build a spatial inverted list for a query location is crucial to extend the threshold algorithm to handle our **MASK** query.

4.2 Index Structure-TGP

Our index is called as the Textual-Grid-Point Inverted Lists (**TGP**) index. The index structure consists of three components.

- Textual Inverted List (**TIL**). The key is a unique keyword in the database D, the value is the list which stores the ids of all points that contain this keyword in D. Similar to the impact-sorted indexes [33], it sorts the lists by a descending order of term weight, and divides the lists into blocks of fixed size. We can determine how many blocks are loaded one time based on the number of points. Let $\mathcal{I}$ denote the inverted list of ψ, each block in $\mathcal{I}$ has a maximum weight $\hat{\mathcal{I}}^{\uparrow}(\psi, it)$ and a minimum weight $\hat{\mathcal{I}}^{\downarrow}(\psi, it)$, where the it is the iteration number of loading, and every list has it_{max} blocks.
- Grid Inverted Lists (**GIL**). The whole space is divided into grids of equal size, and each leaf cell is assigned a unique id using the z-curve cross-coding [40] (see Figure 3). An incremental *round expansion* method [35] is adopted to access the points around the query location, from near to distant. Similar to the bounds of each block in **TIL**, the upper bound of the similarity of unscanned points (along the spatial inverted list) for ρ after it iterations, is denoted by $\hat{\mathcal{G}}^{\uparrow}(\rho, it)$, where $\mathcal{G}$ is the **GIL** for ρ.

  Within each grid cell, we create its associated **TIL**, so we only need to access the objects which have at least one common textual keyword with the query.

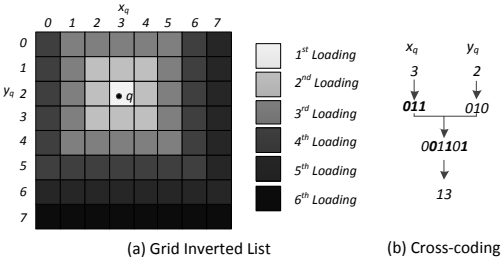

Figure 3: (a) Incremental expansion around the query location within a Grid-index; (b) Labelling a grid cell by Z-curve cross-coding, i.e., picking bit from each binary code successively to form a new code, e.g., bold numbers are from x_q.

- Point Inverted List (**PIL**). Recall Section 2 all existing work such as [30, 35, 42] on top-k trajectory search model a trajectory as a set (or sequence) of points, and their proposed query processing method is built at point granularity. Therefore, we create an inverted index to store a mapping from a point to its corresponding trajectories.

Compressible Index. Our index is composed by inverted lists, each list contains equal number of blocks, and each block stores an array of object ids. Thereby, an array A can be compressed into an array B with fewer integers using the *integer compression* technology [1]. With the corresponding efficient decompression that converts B to A, the query can be conducted without much change, and the performance will not be affected.

4.3 Operators over TGP

By having **TGP** as the index for dataset D, in order to filter objects effectively, we access candidate objects in a batch-wise way from the near to the distant. To access the points and their corresponding objects, we define three main operators.

TextualIterator. Given a query keyword ψ, *TextualIterator* is invoked to get the it-th block from the beginning of textual inverted list **TIL**, as well as the upper bound of the similarity from the query to the unseen objects.

SpatialIterator. Given a query location ρ, in the it-th iteration to access the candidate points, *SpatialIterator* is invoked to get a set of grid-cells in the following way: **1)** get the grid label x_q, y_q on x and y axis of the query location; **2)** compute the x and y value of grid-cells which are going to be scanned; **3)** return the z-curve label by cross-coding as described in Figure 3(b); **4)** find those cells using the labels.

CoveredObjects. Based on the **PIL**, given a set of points, *CoveredObjects* returns all the objects that cover those points. It mainly works for the queries over trajectory data, and can be omitted when dealing with the queries over point data as the object is point itself.

The three operators above can help us achieve the first step of *Threshold Algorithm* in Section 4.1, i.e., scanning the inverted list sequentially. Based on the scanned objects, we will describe how to filter the unseen objects and return top-k result of Step 2 and 3.

4.4 Early Termination & Abandoning over Inverted Lists

As mentioned in our index structure, we expand the search of new candidates from the near to the distant in an incremental way. There are two main challenges: (1) how to stop the expansion as early as possible to reduce the search space; (2) how to efficiently compute the similarity between a candidate and the query. Since we do not store the whole object in the inverted list of a certain term, more random accesses are needed to compute the real similarity between the object and the query. To this end, we propose *early termination* and *early abandoning* on top of our **TGP** to address challenge 1 and 2 respectively. In particular, we propose an algorithm called Early Termination & Abandoning over Inverted Lists-ETAIL, as shown in Algorithm 1.

Early Termination. After several iterations of scanning candidates $R(q_i)$, we can stop expansion based on whether the best of unseen objects can beat all the current top-k result $\mathcal{R}$. More details can be found from Lines 1 to 11. Line 11 shows the condition

for early termination. The bound computation can be found in Section 4.4.1.

Early Abandoning. Before computing the real similarity, we can estimate the upper bound of each candidate's similarity (see Section 4.4.2 for details on upper bound estimation), then determine whether it is bigger than the current $\mathcal{R}$. If so, we will compute the real similarity by accessing the details of each candidate; otherwise, we skip this candidate. Lines 12 to 22 show the details. The terminating condition can be found in the gray area before Line 22.

Algorithm 1: ETAIL Algorithm

Input: Object database D, query Q, k
Output: Top-k result set $\mathcal{R}$

1 $it \leftarrow 1$; $\mathcal{R} \leftarrow \emptyset$; $D_c \leftarrow \emptyset$;
2 **while** $|D_c| < |D|$ **do**
3 **foreach** $q_i \in Q$ **do**
4 $R(q_i) \leftarrow$ TextualIterator$(q_i, it-1, it, TIL)$;
5 $R(q_i) \leftarrow R(q_i) \cup$ SpatialIterator$(q_i, it-1, it, GIL)$;
6 $it \leftarrow it + 1$;
7 $D_c \leftarrow D_c \cup \bigcup_{i=1}^{|Q|}$ CoveredObjects$(R(q_i), PIL)$;
8 **if** $|D_c| > k$ **then**
9 Compute $\hat{S}^{\uparrow}(Q, D - D_c)$ (by Equation 4);
10 Compute $\hat{S}^{\downarrow}(Q, \mathcal{R})$ (by Equation 5);
11 **if** $\hat{S}^{\downarrow}(Q, \mathcal{R}) > \hat{S}^{\uparrow}(Q, D - D_c)$ **then**
12 $seen\_UB[] = \bigcup_{o \in D_c} \hat{S}^{\uparrow}(Q, o)$ (Equation 7);
13 Sort D_c by $seen\_UB[]$ in decreasing order;
14 **foreach** $o_i \in D_c$ **do**
15 Compute $\hat{S}(Q, o_i)$ (Equation 1);
16 **if** $|\mathcal{R}| < k$ **then**
17 Insert o_i in $\mathcal{R}$ and update the order;
18 **else**
19 **if** $\hat{S}(Q, o_i) > \hat{S}(Q, \mathcal{R}[k])$ **then**
20 Replace $\mathcal{R}[k]$ with o_i;
21 **if** $\hat{S}(Q, \mathcal{R}[k]) > seen\_UB[i+1]$ **then**
22 break;
23 **return** $\mathcal{R}$;

4.4.1 Bounds for Early Termination. Here we discuss how to achieve an early termination without scanning the whole database D. Based on our lower bound similarity of the k-th object in the result heap (chosen from the scanned objects D_c). The upper bound of unscanned objects in $D - D_c$ can be computed as:

$$\hat{S}^{\uparrow}(Q, D - D_c) = \alpha \cdot \sum_{\rho_i \in Q_s} \hat{\mathcal{G}}^{\uparrow}(\rho_i, it) + (1-\alpha) \cdot \sum_{\psi_j \in Q_t} \hat{\mathcal{I}}^{\uparrow}(\psi_j, it) \quad (4)$$

where $\hat{\mathcal{G}}^{\uparrow}(\rho_i, it)$ and $\hat{\mathcal{I}}^{\uparrow}(\psi_j, it)$ are the upper bound of unscanned spatial and textual inverted lists for ψ and ρ after it iterations, then the similarity of worst one in the current top-k result can be computed as:

$$\hat{S}^{\downarrow}(Q, \mathcal{R}) = \min_{o \in \mathcal{R}} \hat{S}^{\downarrow}(Q, o) \quad (5)$$

4.4.2 Bounds for Early Abandoning. For each scanned object in D_c, we will compute its upper bound similarity without randomly accessing the missing part, i.e., using less computation to estimate the best case. If the upper bound is smaller than the k-th best result, we will discard it; otherwise, we need to compute the real similarity by accessing the missing part and further check whether it can replace the current k-th result. We choose to sort the upper bounds set $seen\_UB[]$ in line 13 of Algorithm 1, as it enables potential candidates to be scanned earlier, if the current result set is better than the $(i+1)$-th candidate, and all the following candidates are worse with a smaller upper bound, so we can break in line 22.

Particularly, the lower bound for object o can be computed as:

$$\hat{S}^{\downarrow}(Q, o) = \alpha \cdot \sum_{\rho \in Q_s} SProx_{it}^{\downarrow}(\rho, o.\rho) + (1-\alpha) \cdot \sum_{\psi \in Q_t} TRel_{it}^{\downarrow}(\psi, o.\psi) \quad (6)$$

where $SProx_{it}^{\downarrow}(\rho_i, o.\rho) = 0$ if o is not scanned yet in the inverted lists of ρ_i, otherwise $SProx_{it}^{\downarrow}(\rho_i, o.\rho) = SProx(\rho_i, o.\rho)$.

Since the similarities of some terms have not been computed yet, we need to fill the gaps using the maximum similarity of each term. Then the upper bound of o can be computed as:

$$\hat{S}^{\uparrow}(Q, o) = \alpha \cdot \sum_{\rho \in Q_s} SProx_{it}^{\uparrow}(\rho, o.\rho) + (1-\alpha) \cdot \sum_{\psi \in Q_t} TRel_{it}^{\uparrow}(\psi, o.\psi) \quad (7)$$

where $SProx_{it}^{\uparrow}(\rho_i, o.\rho) = \hat{\mathcal{G}}^{\uparrow}(\rho_i, it)$ if o is not scanned yet in the inverted lists of ρ_i, otherwise $SProx_{it}^{\uparrow}(\rho_i, o.\rho) = SProx(\rho_i, o.\rho)$.

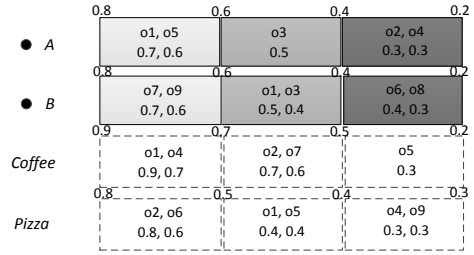

Figure 4: The loaded index structure for ATNN query $Q = \{A, B, \textit{"Coffee"}, \textit{"Pizza"}, 3\}$.

Example 4.1. Here we use an example to illustrate how we achieve early termination and abandoning as shown in Figure 4. Figure 4 shows an ATNN [37] search. Grace is at location A, and wants to meet with friend who is at location B. They choose "Coffee" and "Pizza" as preferences for restaurants to meet. The four partial inverted lists are returned by **TGP**. The two numbers in the upper corners of each block indicate the range of similarities among all objects in this block. The number below each object is the similarity of this object w.r.t. the query.

Early Termination: In the first iteration, we get objects $\{o1, o2, o4, o5, o6, o7, o9\}$. Currently, $\hat{S}^{\uparrow}(D - D_c) = 0.6 + 0.6 + 0.7 + 0.5 = 2.4$. For $o1$, the lower bound $0.7 + 0.9 = 1.6$ which ranks 1 in the top-3 list, another two best are $\{o2 = 0.8, o7 = 0.7\}$, so the $\hat{S}^{\downarrow}(Q, \mathcal{R}) = 0.7 < 2.4$. We keep loading more points. Until that third iteration whose upper bound reduces to $\hat{S}^{\uparrow}(D - D_c) = 0.2 + 0.2 + 0.2 + 0.3 = 0.9$, the lower bound of $o1$ are: $0.7 + 0.5 + 0.9 + 0.4 = 2.5$, then top-3 are: $\{o1 : 2.5, o2 : 1.6, o5 : 1.3\}$. Hence, $\hat{S}^{\downarrow}(Q, \mathcal{R}) = 1.3 > \hat{S}^{\uparrow}(D - D_c) = 0.9$, we terminate.

Early Abandoning: After termination, we have 9 candidate objects: $o1 - o9$. To check whether each of them can be the result,

we compare it with the current top-3 results. For example, for the object $o8$ in the loaded inverted list of location B, its upper bound $\hat{S}^{\uparrow}(Q, o8) = 0.3 + 0.2 + 0.2 + 0.3 = 1 < \hat{S}^{\downarrow}(Q, \mathcal{R}) = 1.3$, thus we can skip $o8$ without further computing its real similarity by accessing the rest unloaded part in the inverted list.

4.5 Discussion on Supporting Boolean Queries

Besides top-k queries with monotone aggregate similarity function, our index **TGP** can also support three kinds of boolean queries which do not have the constraint of k. 1) For a range query, we can find all the grid cells that locate inside or intersect with query range, then access the objects inside the cells and refine it. 2) For a textual boolean query which finds the objects that contain all the keywords of the query, it will be easier as we can fetch the related **TIL** and conduct join operation. 3) For a boolean range query [20] that finds all the objects that contain the query keywords and locate in a query range, we can further access the node-level inverted list in a grid cell after conducting the range query.

5 OPTIMIZATION FOR QUERY REFINEMENT

In this section, we would like to discuss how to optimize the performance of processing continuous multiple queries from a user, which have many overlaps among each other.

5.1 Query Refinement

Definition 5.1. **(Query session)** A query session $\mathcal{S}$ which origins from query Q is composed by a set of queries with continuous refinements on Q.

Generally, the refinements are divided into following three types, which can be combined with each other or itself to compose $\mathcal{S}$.

Definition 5.2. **(Refinement on Keywords)** A query Q is refined to query $Q^{'}$ when a set of keywords $Q_t(\Delta)$ are added into or deleted from the query, denoted as $Q \xrightarrow{Q_t(\Delta)} Q^{'}$.

Definition 5.3. **(Refinement on Points)** A query Q is refined to query $Q^{'}$ when a set of points $Q_s(\Delta)$ are added into or deleted from the query, denoted as $Q \xrightarrow{Q_s(\Delta)} Q^{'}$.

Definition 5.4. **(Refinement on k)** A query Q is refined to query $Q^{'}$ when the user changes the expected number of returned results, denoted as $Q \xrightarrow{k(\Delta)} Q^{'}$.

Figure 5 shows an example for each of the three types of refinements within $\mathcal{S}$. For any two *adjacent queries* Q and Q', there is a high chance that many objects can be candidates for both Q and Q' as they share query locations or (partial) keywords. Therefore, instead of simply invoking Algorithm 1 twice to process two queries independently, we propose to reuse those scanned objects of Q for $Q^{'}$, thereby save computation and IO costs.

5.2 Reusable Scanned Objects

To maintain the scanned objects of Q, we add a new operator called **ReuseBlock**, whose job is to maintain the intermediate inverted lists of query locations and query keywords. For the refined query $Q^{'}$, we can find the shared keywords $Q^{'}_t \cap Q_t$ and locations $Q^{'}_s \cap Q_s$ and

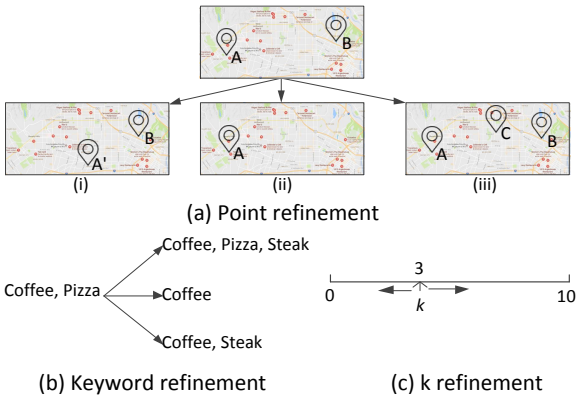

Figure 5: Three kinds of query refinements over geo-tagged data: a) refinement on points, some location(s) in the query i) move, ii) are deleted or iii) are added; b) keywords in the query are i) added, ii) deleted or iii) replaced; c) k is increased or decreased.

reuse their inverted lists without reloading it. Besides the scanned objects of the $Q_t(\Delta)$, we also record the number of iterations to get top-k result of Q, as we may need to conduct more iterations to load more new candidates. We denote the scanned objects and number of iterations as $C(Q).D_c$ and $C(Q).it$ respectively. For the new query keywords and locations $Q_t(\Delta)$ and $Q_s(\Delta)$, we also need to fill the gaps of inverted lists before conducting new expansions for more iterations. We incorporate the optimization above into ETAIL named *Reusable* ETAIL (RETAIL), which supports all three kinds of refinements. Details can be found in Algorithm 2. Next we give a running example of Algorithm 2.

Algorithm 2: RETAIL Algorithm

Input: Trajectory database D, query $Q^{'}$, k, maintained data of former query Q: $C(Q)$
Output: Top-k result set $\mathcal{R}$

1 $Q_t(\Delta) \leftarrow Q^{'}_t - Q^{'}_t \cap Q_t, Q_s(\Delta) \leftarrow Q^{'}_s - Q^{'}_s \cap Q_s$;
2 $it \leftarrow C(Q).it, \mathcal{R} \leftarrow \emptyset, D_c \leftarrow C(Q).D_c$;
3 **while** $|D_c| < |D|$ **do**
4 **foreach** $\psi \in Q_t(\Delta)$ **do**
5 $R_{it}(\psi) \leftarrow$ TextualIterator$(\psi, 0, it, TIL)$;
6 $D_c \leftarrow D_c \cup$ CoveredObjects$(R_{it}(\psi), PIL)$;
7 **foreach** $\rho \in Q_s(\Delta)$ **do**
8 $R_{it}(\rho) \leftarrow$ SpatialIterator$(\rho, 0, it, GIL)$;
9 $D_c \leftarrow D_c \cup$ CoveredObjects$(R_{it}(\rho), PIL)$;
10 Same to Line 8 to 23 of Algorithm 1;
11 $it \leftarrow it + 1$;

Example 5.5. Figure 6 shows two adjacent queries, Q and $Q^{'}$, for the working example of Figure 4. Q takes three iterations to find the top-3 results. Specifically, it has four inverted lists. When a new query $Q^{'}$ comes, since it shares two keywords and one location with Q, we can reuse the inverted lists directly, to avoid loading inverted lists again. As $Q^{'}$ has a new keyword "*Steak*" and a new location $A^{'}$, we will conduct 3 iterations separately based on $it = 3$ in Q to fill the gap of inverted lists (see the black spotted area).

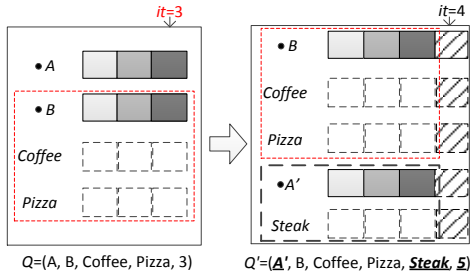

Figure 6: Making loaded objects of Q reusable for an adjacent query Q', the blocks in the red box are reused for Q'.

Table 2: Four datasets for experiments.

	LA [2]	*Twitter* [6]	*GeoLife* [46]	*T-drive* [44]
#Points	215,614	2,267,789	19,476,949	17,511,809
#Keywords	14.67	6.92	N/A	N/A
#Trajectories	31,557	N/A	16,438	10,290
Range	Los Angeles	California & Nevada	Beijing	Beijing

Table 3: Parameter setting.

Parameter	Value
$\|Q\|$: Number of points in Q	1,2,3,4,5,6,7,8,9,10
$\|q.\psi\|$: Number of keywords in query point q	3,4,5,6,7,8,9,10
k: Number of results	10,20,30,40,50,60
α: Similarity weight in Equation 1	0.5
it_{max}: Number of blocks in **TIL** and **GIL**	150

6 EXPERIMENTS

6.1 Settings

Datasets. We take one point dataset and three trajectory datasets. Table 2 shows the statistics of these 4 datasets. Note that *LA* and *Twitter* are two check-in datasets from Foursquare and Twitter, while *LA* can be used as a trajectory dataset by linking multiple check-in points from the same user. *GeoLife* and *T-drive* are two spatial-only trajectory datasets from Beijing, which record the movements of people and taxi respectively, and have much higher sampling rate than the check-in datasets, they are mainly used to test the scalability of our algorithms for their huge amounts.

Query Generation. For each dataset, we generate a query pool randomly according to the range and vocabulary of each dataset. Each query pool contains 1,000 lines and each line is a full query with 10 points and 10 keywords. To generate a query, we can select a full query from the pool first. Then, we reformulate it by deleting a few of last points and keywords according to the setting of parameters $|Q|$ and $|q.\psi|$ in each running. Details about the parameter setting of the query can be found in Table 3, where the default values are underlined. Moreover, all the queries reformulated from a same full query can compose a query session $\mathcal{S}$, then we can also verify our RETAIL proposed in Section 5.

Evaluation Metrics. We explored the running time of all approaches in Section 6.2 and the footprint of their index structures in Section 6.2.4. All experimental results are averaged by running all 1,000 queries. All algorithms are implemented in Java, and run on a PC with an Intel(R) Core(TM) i7-2630QM CPU (8 CPUs) and 8GB RAM & 480GB Kingstom SSD using Windows 10, and our index **TGP** resides in the main memory.

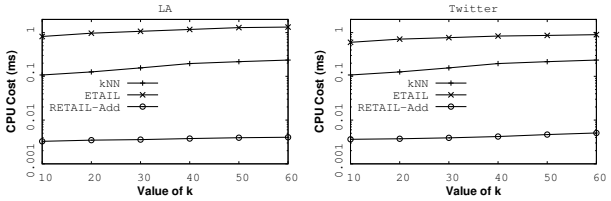

Figure 7: Comparison with [19] by increasing k.

6.2 Performance Evaluation

Before presenting the experimental results on point and trajectory data, we first show our main purposes and the observations.

6.2.1 Purposes and Overall Observations. We will verify:
1) whether our unified algorithm ETAIL **has comparable performance with existing standalone methods**. As our index can support multiple types of queries, we selected kNN [19], ATNN [37] which also covers ANN [24], TkSK [12], kBCT [25] and TkSTT [35] as 5 representative query types, and choose the state-of-the-art (as cited) to compare with ETAIL, R-tree [19, 29] is chosen for kNN as it is the test-of-the-time work. We observe that:

(1) Our **TGP** index can support all 5 types of queries efficiently, and it does not even occupy the largest space.
(2) For three query dependent parameters k, $|q.\psi|$ and $|Q|$, ETAIL can beat most state-of-the-art methods over all 4 datasets with a smaller index size, and by more than one order of magnitude at most, and 5 times in average.
(3) With the increase of k, $|q.\psi|$ and $|Q|$, the running time increases as more candidates need to be scanned.

2) whether our RETAIL **can improve the performance of interactive search.** We add two more groups of experiments for a same query session to verify RETAIL from different perspectives. Specifically, our algorithms are denoted as RETAIL-Add which starts from a simplest query to full query (keep increasing k, number of points or keywords), and RETAIL-Del which starts from a full query to a simplest query (reducing from 10). We observe that:

(1) RETAIL outperforms previous standalone solutions by more than two orders of magnitude at most.
(2) The mutations in the lines of RETAIL in ATNN, TkSK and TkSTT for query refinement is because, too many candidates are inherited from the former long query, so they need more time to check these candidates.

6.2.2 Search over Points. We use *LA* and *Twitter* to evaluate: 1) k Nearest Neighbour (kNN) search by a point. 2) Top-k Spatial Keyword (TkSK) search by a point with $|q.\psi|$ keywords. 3) Aggregate Textual Nearest Neighbour (ATNN) search by $|Q|$ points with $|q.\psi|$ keywords in each query point. We omit the Keyword search (KS) here as our framework is built on top of inverted index, which is widely adopted to answer KS efficiently.

Effect of k on the kNN query. From Figure 7, we find: (1) The running time increases w.r.t. an increasing k. (2) Although R-tree [19] beats ETAIL as we use Grid-index which should be slower than R-tree, it cannot beat our RETAIL with optimization. We do not include RETAIL-Del in the performance study for decreasing k, as we can choose them from existing top-k results directly.

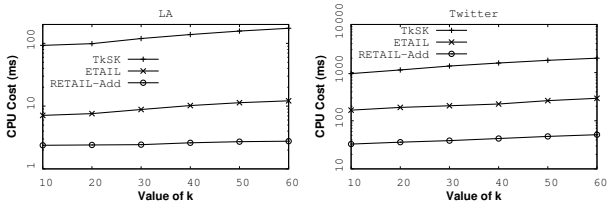

Figure 8: Comparison with [12] by increasing k.

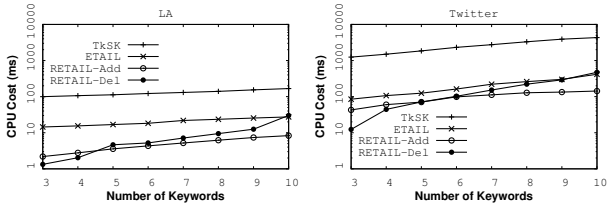

Figure 9: Comparison with [12] by adding keywords.

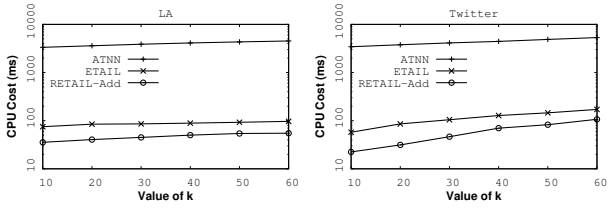

Figure 10: Comparison with [37] by increasing k.

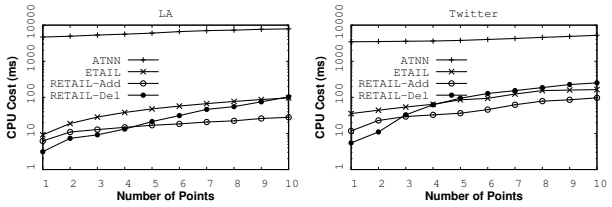

Figure 11: Comparison with [37] by adding points.

Effect of k and $|q.\psi|$ on the TkSK query. From Figures 8 and 9, we find: (1) ETAIL is faster than IR-tree [12] when we process a query without reusing the scanned objects. (2) When treating multiple queries as a query session, RETAIL is more efficient than the aggregated runtime of standalone approach over multiple queries.

Effect of k, $|Q|$ and $|q.\psi|$ on the ATNN query. Figures 10, 11 and 12 show that the running time of all algorithms keep increasing when three parameters rise.

6.2.3 Search over Trajectories. Datasets *LA*, *GeoLife* and *T-drive* are used to evaluate 2 types of queries over trajectory data: 1) k Best Connected Trajectories (kBCT) by $|Q|$ points. 2) Top-k Spatial-textual Trajectory (TkSTT) by $|Q|$ points with $|q.\psi|$ keywords.

Effect of k and $|Q|$ on the kBCT query. Figures 13 and 14 show that only kBCT-GH [34] can compete with ETAIL, while it employs R-tree which occupies more space than **TGP** (see Section 6.2.4). Moreover, RETAIL is 2 orders of magnitude faster than [9].

Effect of $|Q|$ and $|q.\psi|$ on the TkSTT query. Figure 15 shows: (1) the running time of all algorithms keeps increasing when parameters rise, (2) RETAIL-Add is more efficient than RETAIL-Del. We omit the figure of k here as it has a similar trend with other queries.

Statistics of Pruning. Table 4 shows the average number of scanned candidates and pruned blocks in each list over all the candidates of the queries. We focus on two most complex queries: **TkSTT** and **ATNN**. It shows that search over points terminates much earlier than trajectories (highlighted in bold).

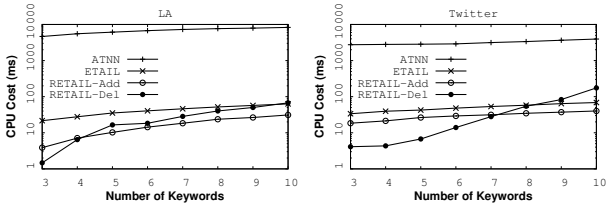

Figure 12: Comparison with [37] by increasing keywords.

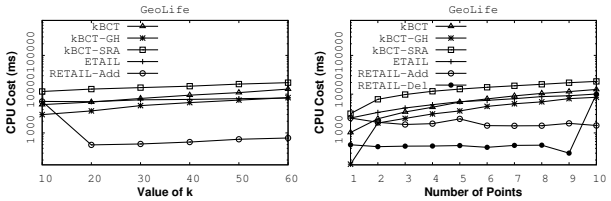

Figure 13: Comparison with [9, 25, 34] over GeoLife dataset.

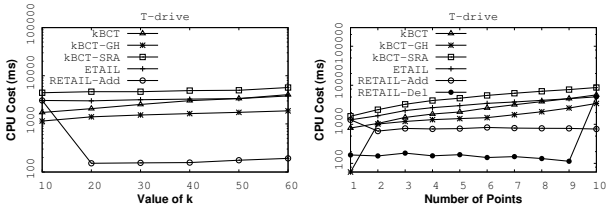

Figure 14: Comparison with [9, 25, 34] over T-drive dataset.

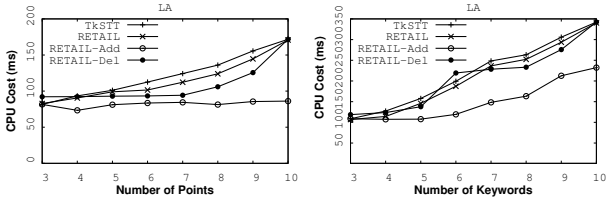

Figure 15: Comparison with [35] by increasing $|Q|$, $|q.\psi|$.

Table 4: Statistics on the pruning power of two kinds of queries over LA dataset. (We adjust $k = 10, 30, 60$, $|Q| = 1, 5, 10$ and $|q.\psi| = 3, 5, 10$ to observe, respectively)

	Parameter	#Scanned Objects			#Pruned Blocks		
ATNN	k	8330	11450	17476	**144**	141	138
	$\|Q\|$	7534	9545	13237	145	144	140
	$\|q.\psi\|$	10004	12650	20235	143	140	135
TkSTT	k	2943	3721	4980	**78**	75	73
	$\|Q\|$	2564	3167	4249	79	75	74
	$\|q.\psi\|$	3254	4035	5132	77	74	72

6.2.4 Index Size. We employ the open-source libraries *FastPFOR*[4] to compress the inverted lists, and *Classmexer*[5] to measure the footprint of our index. From the size of indices in Table 5, we find: (1) Our **TGP** index manage to support all 5 types of queries and even does not occupy the largest space. (2) If we sum up the size of all indices for existing query types together for standalone methods, it will be more than 10 times larger than our **TGP** in *LA* and *Twitter* datasets. (3) For spatial-only datasets like *T-drive* and

[4]https://github.com/lemire/JavaFastPFOR
[5]https://www.javamex.com/classmexer/

GeoLife, **TGP** consumes at least 150MB less than other methods. (4) Our index can be compressed slightly (highlighted in bold).

Table 5: Footprint of indices for different queries.

Query	*LA* [2]	*Twitter* [6]	*GeoLife* [46]	*T-drive* [44]
T*k*STT [35]	261MB	1.5GB	N/A	N/A
T*k*SK [12], ATNN [37]	455.7MB	14.7GB	N/A	N/A
*k*BCT [9], *k*NN [19]	32MB	158MB	1.3GB	1020MB
TGP	89/**2**MB	387/**18**MB	1228.8/**71**MB	810/**52**MB

7 CONCLUSION AND FUTURE WORK

In this paper, we studied how to support various query types over geo-tagged data using a unified indexing and query paradigm. We first concluded the *Monotone Aggregate Spatial Keyword* **MASK** which can represent most existing top-k queries. Then we proposed an index structure called **TGP**, which is composed by grid inverted list, textual inverted list and point inverted list. On top of **TGP** we proposed an Early Termination & Abandoning method over Inverted List (ETAIL) Algorithm which can support multiple query types efficiently. Moreover, RETAIL reuses the scanned objects to improve the performance of interactive searches. In the future, we will further explore a unified paradigm which can support more similarity measures in spatial and textual dimensions.

8 ACKNOWLEDGMENT

This work was supported by ARC DP170102726, DP180102050, and National Natural Science Foundation of China (NSFC) 61728204, 91646204. Zhifeng Bao is supported by a Google Faculty Award.

REFERENCES

[1] Vo Ngoc Anh and Alistair Moffat. 2005. Inverted index compression using word-aligned binary codes. *Information Retrieval* 8, 1 (2005), 151–166.
[2] Jie Bao, Yu Zheng, and Mohamed F Mokbel. 2012. Location-based and preference-aware recommendation using sparse geo-social networking data. In *SIGSPATIAL*. 199–208.
[3] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*. 322–331.
[4] Paul N. Bennett, Filip Radlinski, Ryen W. White, and Emine Yilmaz. 2011. Inferring and using location metadata to personalize web search. In *SIGIR*. 135–144.
[5] Andrei Z Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *CIKM*. 426–434.
[6] Lisi Chen, Gao Cong, Xin Cao, and Kian-lee Tan. 2015. Temporal spatial-keyword top-k publish/subscribe. In *ICDE*. 255–266.
[7] Lisi Chen, Gao Cong, Christian S Jensen, and Dingming Wu. 2013. Spatial keyword query processing: an experimental evaluation. *PVLDB* 6, 3 (2013), 217–228.
[8] Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. 2006. Efficient query processing in geographic web search engines. In *SIGMOD*. 277–288.
[9] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, Yu Zheng, and Xing Xie. 2010. Searching trajectories by locations-an efficiency study. In *SIGMOD*. 255–266.
[10] Zhiyuan Cheng, James Caverlee, Krishna Y Kamath, and Kyumin Lee. 2011. Toward traffic-driven location-based web search. In *CIKM*. 805–814.
[11] Maria Christoforaki, Jinru He, Constantinos Dimopoulos, Alexander Markowetz, and Torsten Suel. 2011. Text vs. space: efficient geo-search query processing. In *CIKM*. 423–432.
[12] Gao Cong, Christian S Jensen, and Dingming Wu. 2009. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB* 2, 1 (2009), 337–348.
[13] Caio Moura Daoud, Edleno Silva de Moura, Andre Carvalho, Altigran Soares da Silva, David Fernandes, and Cristian Rossi. 2016. Fast top-k preserving query processing using two-tier indexes. *Information Processing & Management* 52, 5 (2016), 855–872.
[14] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. 2008. Keyword search on spatial databases. In *ICDE*. 656–665.
[15] Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. 2013. Optimizing top-k document retrieval strategies for block-max indexes. In *WSDM*. 113–122.
[16] Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In *SIGIR*. 993–1002.
[17] Ronald Fagin, Amnon Lotem, and Moni Naor. 2003. Optimal aggregation algorithms for middleware. *J. Comput. System Sci.* 66, 4 (2003), 614–656.
[18] Jiafeng Guo, Gu Xu, Hang Li, and Xueqi Cheng. 2008. A unified and discriminative model for query refinement. In *SIGIR*. 379–386.
[19] Antonin Guttman. 1984. R-trees: a dynamic index structure for spatial searching. *ACM SIGMOD Record* 14, 2 (1984), 47–57.
[20] Ramaswamy Hariharan, Bijit Hore, Li Chen, and Sharad Mehrotra. 2007. Processing spatial-keyword (SK) queries in Geographic Information Retrieval (GIR) systems. In *SSDBM*. 16–25.
[21] Jeff Huang and Efthimis Efthimiadis. 2009. Studying query reformulation strategies in search logs. In *CIKM*. 77–86.
[22] Zhisheng Li, Ken C K Lee, Baihua Zheng, Wang Chien Lee, Dik Lee, and Xufa Wang. 2011. IR-tree: An efficient index for geographic document search. *IEEE Transactions on Knowledge and Data Engineering* 23, 4 (2011), 585–599.
[23] Zhicheng Li, Hu Xu, Yansheng Lu, and Ailing Qian. 2010. Aggregate nearest keyword search in spatial databases. In *APWeb*. 15–21.
[24] Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui. 2005. Aggregate nearest neighbor queries in spatial databases. *ACM Transactions on Database Systems* 30, 2 (2005), 529–576.
[25] Shuyao Qi, Panagiotis Bouros, Dimitris Sacharidis, and Nikos Mamoulis. 2015. Efficient point-based trajectory search. In *SSTD*. 179–196.
[26] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, and Others. 1995. Okapi at TREC-3. *Nist Special Publication Sp* 109 (1995), 109.
[27] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information Processing and Management* 24, 5 (1988), 513–523.
[28] G Salton and M J Mcgill. 1986. *Introduction to modern information retrieval*. McGraw-Hill, Inc., New York, NY, USA.
[29] Timos K Sellis, Nick Roussopoulos, and Christos Faloutsos. 1987. The R+-tree: a dynamic index for multi-dimensional objects. In *VLDB*. 507–518.
[30] Shuo Shang, Ruogu Ding, Bo Yuan, Kexin Xie, Kai Zheng, and Panos Kalnis. 2012. User oriented trajectory search for trip recommendation. In *EDBT*. 156–167.
[31] Fei Song and W Bruce Croft. 1999. A general language model for information retrieval. In *CIKM*. 316–321.
[32] Amanda Spink, Bernard J Jansen, Dietmar Wolfram, and Tefko Saracevic. 2002. From e-sex to e-commerce: Web search changes. *Computer* 35, 3 (2002), 107–109.
[33] Trevor Strohman and W. Bruce Croft. 2007. Efficient document retrieval in main memory. In *SIGIR*. 175–182.
[34] Lu An Tang, Yu Zheng, Xing Xie, Jing Yuan, Xiao Yu, and Jiawei Han. 2011. Retrieving k-nearest neighboring trajectories by a set of point locations. In *SSTD*. 223–241.
[35] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Timos Sellis, Mark Sanderson, and Xiaolin Qin. 2017. Answering top-k exemplar trajectory queries. In *ICDE*. 597–608.
[36] Dingming Wu, Man Lung Yiu, Christian S. Jensen, and Gao Cong. 2011. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*. 541–552.
[37] Kai Yao, Jianjun Li, Guohui Li, and Changyin Luo. 2016. Efficient group top-k spatial keyword query processing. In *APWeb*. 153–165.
[38] Man Lung Yiu, Nikos Mamoulis, and Dimitris Papadias. 2005. Aggregate nearest neighbor queries in road networks. *IEEE Transactions on Knowledge and Data Engineering* 17, 6 (2005), 820–833.
[39] Yong Zeng, Zhifeng Bao, Tok Wang Ling, H. V. Jagadish, and Guoliang Li. 2014. Breaking out of the MisMatch trap. In *ICDE*. 940–951.
[40] Dongxiang Zhang, Chee-Yong Chan, and Kian-Lee Tan. 2014. Processing spatial keyword query as a top-k aggregation query. In *SIGIR*. 355–364.
[41] Dongxiang Zhang, Kian-Lee Tan, and Anthony K. H. Tung. 2013. Scalable top-k spatial keyword search. In *EDBT*. 359.
[42] Kai Zheng, Shuo Shang, Nicholas Jing Yuan, and Yi Yang. 2013. Towards efficient search for activity trajectories. In *ICDE*. 230–241.
[43] Kai Zheng, Han Su, Bolong Zheng, Jiajun Liu, and Xiaofang Zhou. 2015. Interactive top-k spatial keyword queries. In *ICDE*. 423–434.
[44] Yu Zheng. 2011. T-Drive: driving directions based on taxi trajectories. In *SIGSPATIAL*. 99–108.
[45] Yuxin Zheng, Zhifeng Bao, Lidan Shou, and Anthony K H Tung. 2015. INSPIRE : a framework for incremental spatial prefix query relaxation. *IEEE Transactions on Knowledge and Data Engineering* 27, 7 (2015), 1949–1963.
[46] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining interesting locations and travel sequences from GPS trajectories. In *WWW*. 791–800.
[47] Yinghua Zhou, Xing Xie, Chuang Wang, Yuchang Gong, and Wei-Ying Ma. 2005. Hybrid index structures for location-based web search. In *CIKM*. 155–162.
[48] Liang Zhu, Yinan Jing, Weiwei Sun, Dingding Mao, and Peng Liu. 2010. Voronoi-based aggregate nearest neighbor query processing in road networks. In *SIGSPATIAL*. 518–521.