AP-Tree: Efficiently Support Continuous Spatial-Keyword Queries Over Stream

Xiang Wang[†], Ying Zhang[§], Wenjie Zhang[†], Xuemin Lin[†*], Wei Wang[†]
[†]*School of Computer Science and Engineering, University of New South Wales, Australia*
[§]*QCIS, University of Technology Sydney, Australia*
{xiangw, zhangw, lxue, weiw}@cse.unsw.edu.au, Ying.Zhang@uts.edu.au

***Abstract*—We investigate the problem of processing a large amount of continuous spatial-keyword queries over streaming data, which is essential in many applications such as location-based recommendation and advertising, thanks to the proliferation of geo-equipped devices and the ensuing location-based social media applications. For example, a location-based e-coupon system may allow potentially millions of users to register their continuous spatial-keyword queries (e.g., interests in nearby sales) by specifying a set of keywords and a spatial region; the system then delivers each incoming spatial-textual object (e.g., a geo-tagged e-coupon) to all the matched queries (i.e., users) whose spatial and textual requirements are satisfied. While there are several prior approaches aiming at providing efficient query processing techniques for the problem, their approaches belong to *spatial-first* indexing method which cannot well exploit the keyword distribution. In addition, their textual filtering techniques are built upon simple variants of traditional inverted indexes, which do not perform well for the textual constraint imposed by the problem.**

In this paper, we address the above limitations and provide a highly efficient solution based on a novel adaptive index, named AP-Tree. The AP-Tree adaptively groups registered queries using keyword and spatial partitions, guided by a cost model. The AP-Tree also naturally indexes ordered keyword combinations. We present index construction algorithm that seamlessly and effectively integrates keyword and spatial partitions. Consequently, our method adapts well to the underlying spatial and keyword distributions of the data. Our extensive experiments demonstrate that AP-Tree achieves up to an order of magnitude improvement on efficiency compared with prior state-of-the-art methods.

I. Introduction

Due to the proliferation of *User Generated Content* and geo-equipped devices, there is a vast amount of data with both spatial and textual information, referred to as spatial-textual data; they often come in a rapid streaming fashion in many important applications such as social networks (e.g., Facebook, Flickr, FourSquare and Twitter) and location-based services (e.g., location-based advertising). For instance, it is reported that there are about 30 million people sending geo-tagged data out into the Twitterverse, and 2.2 percentage of the global tweets (about 4.4 million tweets a day) provide location data together with the text of their posts.[1]

To make sense of streaming spatial-textual data, it is critical to develop efficient analytical techniques. In this paper, we investigate the problem of continuous spatial-keyword queries over spatial-textual stream, which plays a fundamental role in a variety of applications such as information dissemination [1], location-based recommendation [2] and sponsored search [3].

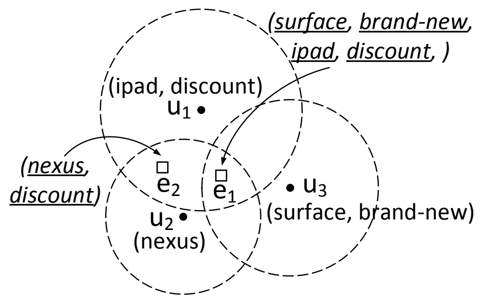
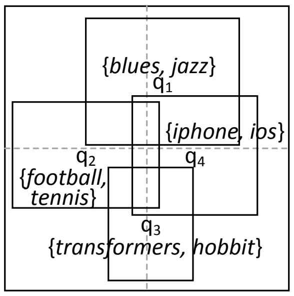


Fig. 1. Location-aware e-coupon system

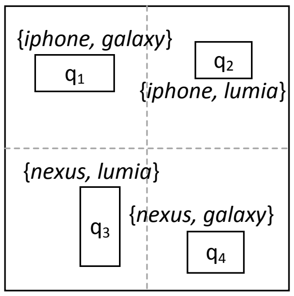
Example 1. *Fig. 1 demonstrates a location-aware publish/subscribe system which delivers e-coupons to potential consumers. A user may register her interest as a query specified by a set of keywords and a spatial region. For instance, user u_1 wants to keep an eye on the discount ipad from nearby shopping malls, and hence issues a query with keywords {ipad, discount} and a circular region as shown in Fig. 1. Suppose two geo-tagged e-coupons e_1 and e_2 are released from two shops. Obviously, a e-coupon* ***matches*** *a query if the e-coupon's location is within the query's region,* and *all the query's search keywords are contained in the e-coupon. Therefore, in this example, e_1 will be delivered to $\{u_1, u_3\}$ and e_2 will be sent to $\{u_2\}$.*

Challenges. There are three key challenges in efficiently processing continuous spatial-keyword queries over spatial-textual streams. *Firstly*, a massive number of queries, typically in the order of millions, are registered in many applications, and hence even a small increase in efficiency results in significant savings. *Secondly*, the streaming spatial-textual objects (e.g., geo-tagged tweets) may continuously arrive in a rapid rate which also calls for high throughput performance for better user satisfaction. *Thirdly*, novel techniques need to be created to develop spatial-textual indexing mechanism that adapts to both the spatial and keyword distributions of the query workload. To the best of our knowledge, [4] and [5] are the only two existing work that systematically study the problem of processing continuous queries over streaming spatial-textual data. Two indexing techniques, IQ-Tree and R^t-Tree, are proposed to match each incoming object to relevant queries following the *filtering-and-refinement* paradigm. Although a large number of irrelevant queries can be pruned by IQ-Tree

*Corresponding author
[1]http://www.futurity.org/tweets-give-info-location



(a) keyword preferred



(b) spatial preferred

Fig. 2. Two motivating examples

and R^t-Tree, they suffer from two fundamental drawbacks.

Firstly, the spatial factor is always prioritized during the index construction regardless of the keyword distribution of the query set. One of our key observations is that the filtering powers based on the spatial and textual constraints may differ substantially under different query workload. Hence, an indexing method must *adapt* to *both* spatial and keyword distributions of the query set to achieve high efficiency. For example, in Fig. 2(a), textual filtering is more effective because regions of the queries are heavily overlapped while queries can be easily distinguished by their keywords. On the contrary, we prefer spatial filtering in Fig. 2(b) since query regions are scattered evenly throughout the space while keywords of the queries are quite similar.

Secondly, the inverted indexing technique adopted in [4], [5] is not well-suited to textual filtering given the nature of the problem is a *superset containment search* [6] from textual perspective. Although inverted indexes have been widely employed in traditional spatial-keyword queries, they are essentially designed for *subset containment queries* [6], where a set of indexed objects *containing* all query keywords are retrieved. We observe that index structures specifically designed for superset containment queries, such as the *ordered keyword trie* [7], shall offer better performance by exploiting the order of keywords, and indexing multiple keyword combinations.

Based on the above observations, we propose a novel index technique, namely the **A**daptive spatial-textual **P**artition **Tree** (AP-Tree for short), to effectively organize continuous spatial-keyword queries. In a nutshell, AP-Tree is a f-ary tree where queries are recursively divided by spatial or keyword partitions (nodes). A cost model is devised to rigorously guide the selection of partition methods such that the construction of the index is adaptive to the query workload. Moreover, we seamlessly and effectively integrate a variant of ordered keyword trie structure [7] to enhance the textual filtering performance. Extensive experiments show that our AP-Tree based matching algorithm achieves very substantial improvements over all previous studies due to its adaptiveness. For instance, with 20 million registered queries, our method can process around $2,500$ tweets per second, compared with about 300 tweets by the previous methods.

Contributions. The principle contributions of this paper are summarized as follows.

- We devise a novel adaptive spatial-textual partition tree (AP-Tree for short) to tackle the problem of continuous spatial-keyword queries over streaming spatial-textual objects. To the best of our knowledge, this is the first spatial-textual indexing mechanism which adaptively prioritizes spatial and keyword partition methods.
- A cost model is proposed to evaluate the goodness of *keyword partition* and *spatial partition*. For *keyword partition*, an optimal algorithm as well as an efficient heuristic algorithm are devised. As to *spatial partition*, we show that finding optimal spatial partition is *NP-hard*, and propose an efficient heuristic algorithm instead. With the guide of cost model, AP-Tree is constructed in an adaptive way to minimize overall cost. Moreover, we show that AP-Tree is self-adjustable to the change of query workload.
- Comprehensive experiments show that our new matching algorithm achieves substantial improvements (up to an order of magnitude speed up) over the state-of-the-art techniques.

TABLE I
THE SUMMARY OF NOTATIONS

Notation	Definition
o	a spatial-textual object
q	a continuous spatial-keyword query
$o.\psi$ ($q.\psi$)	a set of keywords for object o (query q)
$o.loc$ ($q.r$)	object location (query region)
w, w_i, w_j	keyword (term)
$\mathcal{Q}$ (Q)	query set (subset of $\mathcal{Q}$)
$\mathcal{O}$ (O)	object stream (subset of $\mathcal{O}$)
$\mathcal{V}$ (V)	vocabulary (subset of $\mathcal{V}$)
N	a node of AP-Tree
N_l	offset of node N
N_r	spatial region of node N
f	fanout of AP-Tree node
θ_q	partition termination threshold
θ_{KL}	KL-Divergence threshold

Roadmap. The rest of this paper is organized as follows. We introduce preliminary and related work in Section II. Section III presents the framework of AP-Tree and matching algorithm. The cost model and index construction algorithms are devised in Section IV. Extensive experiments are depicted in Section V. Finally, Section VI concludes the paper.

II. PRELIMINARIES

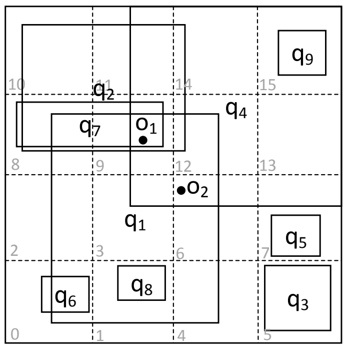
We first formally define the problem of continuous spatial-keyword queries in Section II-A. Section II-B reviews related work, and Section II-C presents two state-of-the-art techniques, IQ-Tree [4] and R^t-Tree [5]. Section II-D introduces the ordered keyword trie [7]. Table I summarizes the mathematical notations used throughout this paper.

A. Problem Definition

In this paper, $\mathcal{O}$ denotes a sequence of streaming spatial-textual objects. A spatial-textual object is a textual message with geo-location, such as check-ins and geo-tagged tweets. Formally, a spatial-textual object o is modeled as $o = (\psi, loc)$, where $o.\psi$ denotes a set of distinct terms (keywords) from a vocabulary set $\mathcal{V}$ and $o.loc$ represents a geo-location [2].

Definition 1 (Continuous Spatial-keyword Query)**.** *A continuous spatial-keyword query q is defined as $q = (\psi, r)$, where*

[2]We assume the location of object is a point while our techniques can be immediately extended to support a spatial region.



q	ψ
q_1	w_1,w_2
q_2	w_1,w_2
q_3	w_2,w_4
q_4	w_1,w_5
q_5	w_1,w_3
q_6	w_3,w_5
q_7	w_3,w_4
q_8	w_4,w_6
q_9	w_5,w_6

o_1:ψ ={w_2,w_3,w_4}
o_2:ψ ={w_1,w_2,w_5}

Fig. 3. Running example

$q.\psi$ is a set of distinct user-specified keywords, and $q.r$ is a spatial region (e.g., a rectangle or a circle).

Note that a continuous spatial-keyword query is a long-running query, and is valid until it is deregistered. We say an incoming spatial-textual object *matches* (or *satisfies*) a continuous spatial-keyword query if it satisfies both spatial and keyword constraints of the query. Following is a formal definition.

Definition 2 (Matching). *A spatial-textual object matches a continuous spatial-keyword query if and only if the following two conditions are satisfied: (1) $o.\psi \supseteq q.\psi$, and (2) $o.loc \in q.r$.*

Problem Statement. In this paper, we tackle the problem of continuous spatial-keyword queries against streaming spatial-textual data. Specifically, given a set $\mathcal{Q}$ of continuous queries, for each incoming object o from streaming spatial-textual data $\mathcal{O}$, we aim to rapidly deliver o to all the *matched* queries.

Example 2. *Fig. 3 depicts a running example used throughout this paper. In this example, there are 9 registered queries $\{q_1, \dots, q_9\}$ and two objects $\{o_1, o_2\}$. Specifically, o_1 falls in the search regions of $\{q_1, q_2, q_4, q_7\}$, and its keywords only fully contain all the keywords of q_7. Thus, o_1 is delivered to $\{q_7\}$. With similar rationale, o_2 matches queries $\{q_1, q_4\}$.*

In the paper hereafter, we abbreviate the spatial-textual object and the continuous spatial-keyword query as *object* and *query* respectively, if there is no ambiguity. We assume there is a total order for keywords in $\mathcal{V}$, and the keywords in each query and object are sorted accordingly. For presentation simplicity, we assume $w_i < w_j$ if $i < j$.

B. Related Work

Spatial-keyword Search. In recent years, spatial-keyword search has attracted great attention, which aims to retrieve the *relevant* spatial-textual objects for a given spatial-keyword query. Existing work usually combines keyword indexing and spatial indexing techniques to organize objects such that non-promising objects can be quickly pruned from both spatial and textual perspectives. In general, these techniques can be classified into two categories: *keyword-first* (e.g., [8], [9], [10], [11]) and *spatial-first* (e.g., [12], [13]). Note that a spatial-keyword search is an *ad-hoc* query (i.e., user-initiated model) while our problem focuses on *continuous* queries (i.e., server-initiated model).

Continuous Query Processing. In continuous query processing system, there are many long-running queries on the server side. The incoming objects are continuously evaluated and reported to the *matched* queries registered at the server. Many studies on publish/subscribe systems investigate a variety of continuous queries such as predicate-based matching (e.g., [14], [15], [16], [17]) and similarity-based ranking (e.g., [18], [19]). Nevertheless, they do not consider the spatial information. Recently, continuous moving spatial-keyword queries have been investigated (e.g., [20], [21]), but they focus on continuously reporting relevant objects where queries are moving, which are inherently different from our problem. There are some existing work on the location-aware publish/subscribe systems, but most of them either cannot properly handle large scale streaming data (e.g., [22]) or do not consider the textual information (e.g., [23]). Another recent work [24] proposes a temporal publish/subscribe system considering both spatial and keyword factors. However, its semantics (i.e., top-k matching) are different from ours (i.e., boolean filtering). To the best of our knowledge, [4] and [5] are only two existing work systematically studying the same problem of continuous queries over streaming spatial-textual data as ours, which are introduced in Section II-C.

C. IQ-Tree and R^t-Tree

Very recently, [4] and [5] independently investigate the problem of continuous queries over streaming spatial-textual data. Two efficient indexing techniques, namely IQ-Tree and R^t-Tree, are proposed to organize a massive number of queries. Both IQ-Tree and R^t-Tree belong to *spatial-first* indexing mechanism where spatial feature is preferred during index construction.

In IQ-Tree [4], queries are organized by a Quadtree where each query is attached to one or multiple Quadtree cells according to a cost model which aims to balance matching and update costs. For each cell, the related queries are organized by a ranked-key inverted list [25], and a query is assigned to the posting list of its least frequent keyword. Fig. 4 shows an example of IQ-Tree where 9 queries in the running example are organized. In particular, query q_7 in cell 9 is in the posting list of w_4 since w_4 is the least frequent keyword among $q_7.\psi = \{w_3, w_4\}$. The matching algorithm of IQ-Tree follows the *filtering-and-refinement* paradigm. For instance, regarding the incoming object o_1 in Fig. 3, unpromising queries are first pruned based on their search regions, i.e., only queries which reside on the cells penetrated by o_1 (gray cells) survive the spatial filtering. Then keyword filtering is applied, and only the queries on the posting lists of the object keywords are retrieved, which correspond to $\{q_1, q_2, q_7\}$. Finally, candidate queries are verified based on their search regions and query keywords and object o_1 is delivered to query $\{q_7\}$. The total number of queries verified in this example is 3.

Regarding R^t-Tree [5], queries are indexed by an R-Tree based on their search regions. Each R-Tree node also records the keywords of its descendant queries, namely *token filter*,

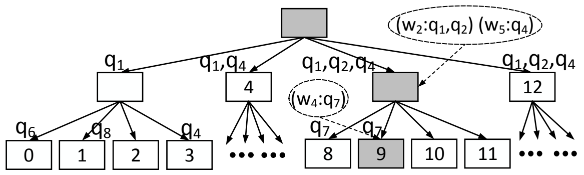


Fig. 4. Example of IQ-Tree

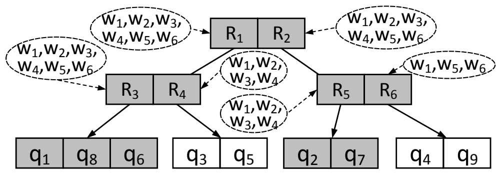


Fig. 5. Example of R^t-Tree

for textual filtering purpose. Two variants of R^t-Tree, namely R^{t+}-Tree and R^{t++}-Tree, further improve the performance by carefully choosing one and multiple representative tokens (keywords), respectively. Fig. 5 demonstrates an example of R^t-Tree on the running example. At each node, it employs both spatial and keyword filtering techniques to prune unpromising queries. For example, to match object o_1 in Fig. 3, we need access all the gray nodes, and verify $\{q_1, q_8, q_6, q_2, q_7\}$ in the leaf nodes according to spatial and keyword constraints. Note that R_6 is pruned because its token set, i.e., $\{w_1, w_5, w_6\}$, has no overlap with keywords in o_1, i.e., $\{w_2, w_3, w_4\}$, while R_4 is pruned by spatial constraint. The total number of queries verified in this example is 5.

D. Ordered Keyword Trie

The problem of *superset containment search* has been extensively studied in the literature and a variety of efficient techniques are proposed (e.g., [3], [26], [7]). Specifically, given a set of queries and an object, each of which consists of a set of keywords, we aim to find queries whose query keywords are fully contained by the object keywords. Clearly, the nature of our problem is a superset containment search if the spatial dimension is not considered.

To efficiently support *superset containment search*, Zeinab *et al.* [7] recently propose an ordered keyword trie structure where each node corresponds to a keyword assuming there is a global order for all keywords. Each query is indexed based on its ordered query keywords (i.e., "prefixes"). Fig. 6 depicts the ordered keyword trie structure over the running example where each query can be accessed through a unique path following its ordered keywords. For instance, q_1 with keywords $\{w_1, w_2\}$ can be visited through the path as indicated by the dotted polygon. Given object o_1 with $o_1.\psi = \{w_2, w_3, w_4\}$, we only need to visit gray nodes in Fig. 6 and come up with final matches $\{q_3, q_7\}$ w.r.t. keywords only.

In this paper, we integrate a variant of the ordered keyword trie structure in AP-Tree to efficiently support textual filtering.

III. AP-Tree Framework

In this section, we present a novel adaptive spatial-textual indexing mechanism to organize continuous queries, namely

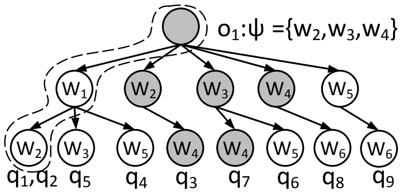


Fig. 6. Example of ordered keyword trie

AP-Tree (**A**daptive **P**artition Tree). Section III-A introduces the motivation of the AP-Tree. Section III-B describes the AP-Tree structure, followed by a detailed matching algorithm in Section III-C.

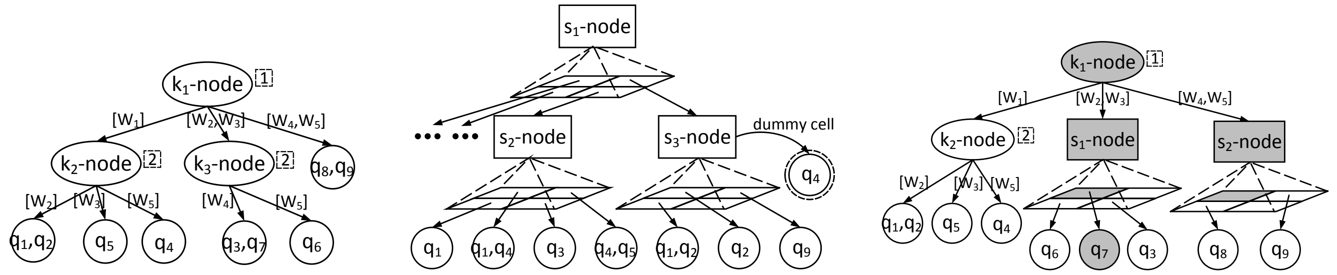
A. Motivation

Due to the massive number of continuous queries, it is imperative to devise efficient indexing technique such that a large number of unpromising queries can be filtered at a cheap cost. We show that a good indexing mechanism over continuous spatial-keyword queries should satisfy following three criteria.

(1) Adaptiveness. Intuitively, with respect to different keyword and location distributions of the query workload, both spatial feature and textual feature may become the dominant factor. This observation is illustrated in Fig. 2, and substantiated by our empirical study. As shown in Section II-C, tree structure of IQ-Tree [4] and R^t-Tree [5] is only determined by the spatial feature. Although the keyword filtering component (e.g., local inverted list) is augmented to tree nodes, their overall performance is unavoidably deteriorated. On the other hand, our experiments show that textual dominant indexing approach also suffers from the same problem. This motivates us to devise a novel textual and spatial partition based f-ary tree structure so that the queries are indexed in an adaptive and flexible way w.r.t. the query workload. Moreover, the index should be self-adjustable to the change of query workload. In particular, two types of partition strategies, namely *keyword partition* and *spatial partition*, are proposed to recursively partition a set of queries by textual feature and spatial feature, respectively. A node partitioned by textual (resp. spatial) feature is called *keyword* (resp. *spatial*) *node*. A cost model (Section IV-A) is developed to decide which partition approach is employed at each node.

(2) Efficient Keyword Filtering. From textual perspective, our problem is essentially a *superset containment search*; that is, finding queries whose keywords are *fully contained* by a given object. Among existing techniques (e.g., [3], [26], [7]), ordered keyword trie [7] demonstrates its superior performance because it takes great advantage of common prefixes of the ordered query keywords. Moreover, it is a hierarchical tree structure. This motivates us to integrate the ordered keyword trie for keyword filtering purpose. To accommodate the f-ary tree structure of AP-Tree, we partition related keywords on the tree node into f parts based on our cost model, instead of keeping each individual keyword[3]. Optimal and heuristic keyword partition methods are proposed in Section IV-B.

[3]Note that there are about 1.7 million distinct keywords in the tweet dataset.



(a) *keyword partition* (b) *spatial partition* (c) AP-Tree

Fig. 7. Examples of *keyword partition*, *spatial partition* and AP-Tree

(3) Efficient Spatial Filtering. Regarding the spatial filtering, our problem corresponds to the *point stabbing search* [27] in 2-dimensional space; that is, identifying query rectangles which are stabbed by the geo-location of the incoming object. The best known data structure for the point stabbing problem is the segment tree [28] which can retrieve all k related rectangles with search time $O(\log(m) + k)$ where m is the number of queries. However, segment tree is not well-suited to large scale data because the space usage of $O(m \log m)$ on 2-dimensional data. *Space-oriented* (e.g., Quadtree) and *object-oriented* (e.g., R-Tree) partition strategies are adopted in [4] and [5] respectively due to their good support of point stabbing search and scalability. As stressed in [4], *space-oriented* partition strategy is more suitable to spatial filtering because of its disjoint space decomposition policy and good support of query regions with different sizes. Our empirical study also substantiates this claim. Motivated by this, we adopt *space-oriented* partition approach for *spatial partition*. In particular, the region of each spatial node is partitioned into f grid cells guided by the cost model. As it is an NP-hard problem to find optimal spatial partition, an efficient heuristic algorithm is designed in Section IV-C.

B. AP-Tree Structure

Based on the above motivations, we devise an adaptive spatial-textual partition tree (AP-Tree for short) which employs *keyword partition* and *spatial partition* methods to recursively divide queries in a top-down manner. In this paper, N denotes an AP-Tree node and there are three types of nodes: *keyword node* (k-node), *spatial node* (s-node), and *query node* (q-node). An intermediate node is a keyword (resp. spatial) node if *keyword partition* (resp. *spatial partition*) is adopted. We use f to denote the fanout of the intermediate node. A leaf node of AP-Tree corresponds to a q-node, and each query will be assigned to one or multiple query nodes according to its query region and ordered query keywords.

Below, we introduce *keyword node* and *spatial node* in details.

Keyword Node. We assume there is a total order among keywords in the vocabulary $\mathcal{V}$, and keywords in each object and query are sorted accordingly. We delay the discussion of the effect of keyword order strategy to the experimental part. Queries assigned to a node N are partitioned into f *ordered cuts* according to their N_l-th keywords, where N_l is called the *partition offset* of the node N. We have $N_l \leq N_l^*$ if N^* is a descendant keyword node of N. An *ordered cut* is an interval of the ordered keywords, denoted as $c[w_i, w_j]$, where w_i and w_j ($w_i \leq w_j$) are boundary keywords. For presentation simplicity, we use $c[w_i]$ to denote $c[w_i, w_i]$ if there is only one keyword in the cut.

Example 3. *Fig. 7(a) shows a special case of AP-Tree in which only keyword partition is employed on the running example. We use an oval to represent a k-node and the number on its right side indicates the partition offset. Meanwhile, a q-node is denoted by a circle. Assume there are at most* 3 *ordered cuts on each keyword node. In* k_1*-node with partition offset* 1*, we collect the first keywords of* 9 *queries which correspond to* $\{w_1, w_2, w_3, w_4, w_5\}$*. These keywords can be divided into* 3 *cuts:* $c[w_1]$*,* $c[w_2, w_3]$ *and* $c[w_4, w_5]$*. Queries* $\{q_1, q_2, q_4, q_5\}$ *are assigned to* $c[w_1]$ *whose corresponding node is* k_2*-node. Since the partition offset of* k_2*-node is* 2*, the second keywords of these queries, i.e.,* $\{w_2, w_3, w_5\}$*, are used to assign queries into three cuts:* $c[w_2]$*,* $c[w_3]$ *and* $c[w_5]$*, each of which is associated with a q-node.*

Spatial Node. The space is recursively partitioned by spatial nodes. Let N_r denote the region of a spatial node N, which will be divided into f grid cells. A query on a spatial node N is pushed to a grid cell c if $q.r$ overlaps c or contains c. Note that, unlike the keyword node in which a query is assigned to an unique cut, a spatial node may assign a query to multiple cells.

Example 4. *Fig. 7(b) depicts another special case of AP-Tree in which only spatial partition is employed on the running example. Here, we use a rectangle to represent a s-node. In each spatial node, the spatial region is partitioned into* 4 *cells. To match an object, we simply navigate through the spatial nodes which contain the object location, until we reach the leaf node. We remark that the cells on each spatial node may not be of equal size.*

For each keyword node N, a query q assigned to N cannot find a cut if there is no enough query keywords, i.e., $|q.\psi| < N_l$. We use a *dummy cut* to keep these queries. Similarly, each spatial node N has a *dummy cell* for the queries which contain the region of N (i.e., $N_r \subseteq q.r$) and hence do not need to be further partitioned on node N. Note that queries on the dummy cut (resp. cell) may be further partitioned by spatial (resp. keyword) node only, or simply maintained by a query node. For instance, the node indicated by dotted circle in

Fig. 7(b) is actually a dummy node, because the query region of q_4 fully contains the region of s_3-node.

Example 5. *Fig. 7(c) illustrates an example of AP-Tree constructed over the running example, where both keyword and spatial partitions are employed. Queries are recursively partitioned by keyword nodes or spatial nodes, and finally assigned to query nodes.*

C. Object Matching

In this subsection, we present efficient AP-Tree based object matching algorithm. Following the *filtering-and-verification* paradigm, we navigate through AP-Tree to prune non-promising queries by utilizing spatial or keyword filtering techniques, and then verify the candidate queries on query nodes accessed.

Algorithm 1 depicts the procedure to retrieve all the *matched* queries for a given object o. It is a recursive procedure invoked by each accessed intermediate node with a depth-first search strategy. In particular, we simply verify the associated queries if a q-node is accessed, and matched queries are kept in $\mathcal{R}$ (Line 2). Regarding s-node (Lines 12-15), we only need to access the cell c stabbed by o (i.e., $o.loc \in c_r$) as well as the dummy cell. Recall that the dummy cell of a s-node keeps queries covering the region of the node, and may be further partitioned by keyword node only. As to the k-node (Lines 5-10), let w_1, w_2, ..., $w_{|o.\psi|}$ denote all the object keywords in $o.\psi$. For each k-node N accessed, we use η to denote the start matching position regarding the object keywords. Line 6 identifies the corresponding cut for each object keyword w_j ($\eta \leq j \leq |o.\psi|$). For each cut hit by at least one object keyword, we further explore its corresponding node at Line 8 where η is set to $i+1$ and w_i denotes the smallest keyword which hits the cut. Similar to s-node, dummy cut will be explored (Line 10) since all queries on the dummy cut survive the keyword filtering according to its definition. For each incoming object o, we retrieve all the *matched* queries by calling the function **ObjectMatching**(o, 1, $root$), where $root$ is the root node of AP-Tree.

Example 6. *Suppose* 9 *queries in the running example (Fig. 3) are organized by AP-Tree as shown in Fig. 7(c). For the incoming object* o_1*, we first access* k_1*-node with* $\eta = 1$*. According to Lines 5-10, the cut* $c[w_2, w_3]$ *on* k_1*-node is hit by the first and second object keyword* w_2 *and* w_3 *in* o_1*. Therefore,* s_1*-node will be explored with* $\eta = 1 + 1 = 2$*. Similarly,* s_2*-node is accessed with* $\eta = 3 + 1 = 4$*. Regarding* s_1*-node, we identify the grid cell stabbed by* $o_1.loc$ *(shaded cell on* s_1*-node), and reach the corresponding query node, which contains* $\{q_7\}$*. We verify* q_7 *and put it into* $\mathcal{R}$ *because it satisfies both keyword and spatial constraints. The same procedure is applied to* s_2*-node. Since there is no q-node on the cell stabbed by* o_1 *(shaded cell on* s_2*-node), none of the q-nodes of* s_2*-node will be accessed. Finally, we have* $\mathcal{R} = \{q_7\}$*. In this example, the total number of queries verified is only* 1.

Time Complexity. The dominant cost of Algorithm 1 is the AP-Tree traverse cost and verification cost. The traverse costs are $O(|o.\psi| \times \log(f))$ and $O(\log(f))$ for each k-node and

Algorithm 1: **ObjectMatching**(o, η, N)

```
Input  : $o$ : incoming object
         $\eta$ : the start matching position regarding $o.\psi$
         $N$ : node accessed currently
Output : $\mathcal{R}$ : set of all the matched queries
1  if $N$ is a $q$-node then
2      Verify queries in $N$ and insert the matched ones to $\mathcal{R}$ ;
3      return
4  if $N$ is a $k$-node then
5      for $\eta \leq i \leq |o.\psi|$ do
6          Find the corresponding $cut$ based on $w_i$ in $o.\psi$;
7          if $cut$ has not been visited then
8              ObjectMatching$(o, i+1, cut)$ ;
9      if $dummy\_cut$ exists then
10         ObjectMatching$(o, \eta, dummy\_cut)$;
11 else
12     Find the $cell$ which covers $o.loc$ using grid structure;
13     ObjectMatching$(o, \eta, cell)$;
14     if $dummy\_cell$ exists then
15         ObjectMatching$(o, \eta, dummy\_cell)$;
```

s-node, respectively. The verification cost of a query q is $O(|o.\psi| + |q.\psi|)$ in the worst case, while the number of verifications heavily depends on the filtering capacity of AP-Tree.

Algorithm Correctness. Since each query will be validated at Line 2, it is immediate that all queries in $\mathcal{R}$ are valid. As a query may be assigned to disjoint grid cells at each spatial node and the union of these cells contains the query region, each matched query q must be assigned to a query node whose ancestor spatial nodes are stabbed by the object location. Let $B_1, B_2, \ldots, B_m$ denote the buckets (cuts or cells) along the path from $root$ of AP-Tree to this query node. It is immediate that cell B_1 will be visited if the root is a s-node. Similarly, the cut B_1 will be visited if the root is a k-node since there must exist one object keyword which is equal to the first query keyword of q. It is easy to see that B_i will be visited sequentially for $1 < i \leq m$, and the correctness of Algorithm 1 follows.

IV. AP-Tree Construction and Maintenance

We first propose a cost model in Section IV-A to quantitatively analyze the goodness of keyword and spatial partitions. Then efficient keyword and spatial partition approaches are devised to minimize the matching cost in Section IV-B and Section IV-C, respectively. Section IV-D presents the AP-Tree construction algorithm which adaptively selects keyword and spatial partition methods to construct AP-Tree in a top-down manner. Section IV-E develop dynamic maintenance approach which makes AP-Tree self-adjustable to the change of query workload.

A. Cost Model

Given a set $\mathcal{Q}$ of queries, AP-Tree is constructed in a top-down manner. Thus, we need to evaluate the goodness of a keyword or spatial partition such that the AP-Tree is adaptive to query workload. In this subsection, we propose a cost model to quantitatively measure the matching cost for two partition methods. Given a node N and a set Q of queries assigned to N, without further partition the matching cost contributed by N is $|Q|$ assuming the average query verification cost is a unit time. Clearly, we can partition $|Q|$ queries into a set $\mathcal{P}$ of f buckets by *keyword partition* or *spatial partition* to reduce the matching cost. Throughout this paper, we might use *bucket* and *cut*, *bucket* and *cell* interchangeably for better understanding of the idea.

Let B denote a bucket of the partition, we use $\mathbf{w}(B)$ to record its *weight* which is the number of queries associated to B. By $\mathbf{p}(B)$ we mean the *hit probability* of the bucket B, i.e., the probability that B is explored during the object matching. The expected matching cost regarding partition $\mathcal{P}$, denoted by $C(\mathcal{P})$, is as follows.

$$C(\mathcal{P}) = \sum_{i=1}^{f} \mathbf{w}(B_i) \times \mathbf{p}(B_i) \qquad (1)$$

Given a partition $\mathcal{P}$ and a set of queries Q on the node, the calculation of $\mathbf{w}(B)$ is immediate for each bucket B. We may derive the hit probability $\mathbf{p}(B)$ based on some distribution assumptions or object workload. For analysis simplicity, we assume that $\mathbf{p}(B) = \sum_{w \in B} \mathbf{p}(w)$ for keyword node, where $\mathbf{p}(w)$ is the *hit probability* of the keyword w. In case a set O of the objects is available, it is trivial to derive hit probability of each individual keyword. Otherwise, we assume the query keyword with high frequency among Q has better chance to appear in object keywords; that is, we use query workload to simulate object workload. Specifically, we set $\mathbf{p}(w) = \frac{freq(w)}{\sum_{w \in \mathcal{P}} freq(w)}$ where $freq(w)$ is the frequency of keyword w among all queries in Q. Regarding *spatial partition*, we may simply assume the uniform distribution of the object location, and hence $\mathbf{p}(B) = \frac{Area(B)}{Area(N)}$ where $Area(B)$ is the area of the bucket (i.e., cell) B and $Area(N)$ is the region size of the node N. The hit probability calculation of each cell (bucket) is immediate when object workload is available.

B. Keyword Partition

Without loss of generality, we assume the l-th keywords of the queries in Q correspond to a set of *ordered* keywords $V = \{w_1, w_2, \dots, w_{|V|}\}$. On each keyword node, queries are partitioned into f *ordered cuts* based on their l-th keywords, and we aim to find an optimal keyword partition, denoted by $\mathcal{P}_k^*$, such that the matching cost is minimized. We first present a dynamic programming approach to achieve the optimal partition, followed by a simple optimal solution for a special case. Then we develop an efficient heuristic approach.

(1) Optimal Partition.

Dynamic Programming Algorithm. By $\mathcal{P}_k(i,j,c)$ we mean a keyword partition regarding keywords between w_i and w_j (both inclusive) with c cuts. The *optimal partition* is denoted by $\mathcal{P}_k^*(i,j,c)$. Since keywords are ordered, we can come up with $\mathcal{P}_k^*(i,j,c)$ by exhausting all possible locations of the first cut as follows.

$$\begin{aligned} C(\mathcal{P}_k^*(i,j,c)) \quad = \quad & \min_{i \le m \le j-c+1} (\ C(\mathcal{P}_k^*(i,m,1)) + \\ & C(\mathcal{P}_k^*(m+1,j,c-1)) \ ) \end{aligned} \qquad (2)$$

Let $\mathcal{P}_k^*(i,m,1)$ represent the optimal partition which consists of one cut $c[w_i, w_m]$, we have

$$C(\mathcal{P}_k^*(i,m,1)) = (\sum_{j=i}^{m} \mathbf{w}(w_j)) \times (\sum_{j=i}^{m} \mathbf{p}(w_j)) \qquad (3)$$

where $\mathbf{w}(w_j)$ denotes the number of queries whose l-th keyword equals w_j.

Algorithm 2 illustrates our dynamic programming method for optimal keyword partition. In particular, Lines 1-2 compute the cost for each partition with single cut. Then Lines 3-5 iteratively compute the optimal partitions with c cuts ($2 \le c \le f-1$). Finally, the optimal keyword partition $\mathcal{P}_k^*$ corresponds to $\mathcal{P}_k^*(1,|V|,f)$. The time complexity of Algorithm 2 is $O(f \times |V|^2)$.

Algorithm 2: **Optimal Keyword Partition**(V, f)

```
  Input  : $V$ : keyword set to be partitioned
           $f$ : number of cuts
  Output : $\mathcal{P}_k^*$ : optimal keyword partition
1 for $1 \le i \le j \le |V|$ do
2     Compute $C(\mathcal{P}_k^*(i,j,1))$ based on Equation 3 ;
3 for $2 \le c \le f-1$ do
4     for $1 \le i \le |V|+1-c$ do
5         Compute $C(\mathcal{P}_k^*(i,|V|,c))$ based on Equation 2 ;
6 Compute $C(\mathcal{P}_k^*(1,|V|,f))$ based on Equation 2 ;
7 return $\mathcal{P}_k^*(1,|V|,f)$
```

Optimal solution for special case. We say the query workload and object workload have similar distribution if and only if $\frac{\mathbf{p}(w_i)}{\mathbf{w}(w_i)} = \lambda$ for any $1 \le i \le |V|$. In this special case, we come up with a simple optimal solution with time $O(|V|)$ if each cut has the same weight. In particular, the cost model in Equation 1 now turns to

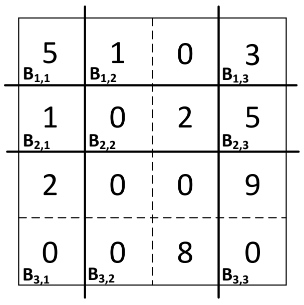
$$C(\mathcal{P}) = \lambda \sum_{i=1}^{f} \mathbf{w}(B_i)^2 \qquad (4)$$

According to Cauchy-Schwarz Inequality, we have $(\sum_{i=1}^{f} \mathbf{w}(B_i)^2)(\sum_{i=1}^{f} 1^2) \ge (\sum_{i=1}^{f} \mathbf{w}(B_i) \times 1)^2$. Therefore, $C(\mathcal{P})$ can achieve the optimal solution if $\mathbf{w}(B_i) = \mathbf{w}(B_j)$ for $1 \le i,j \le f$. Note that as discussed in Section IV-A, we use query workload to simulate object workload when object workload is unavailable, and hence two distributions are similar.

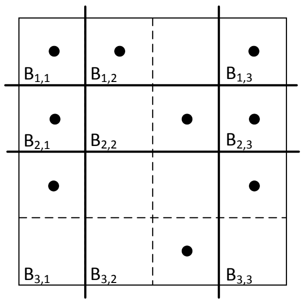
Algorithm 3: **Heuristic Keyword Partition**(V, f)

```
  Input  : $V$ : keyword set to be partitioned
           $f$ : number of cuts
  Output : $\mathcal{P}_k$ : keyword partition
1 Find a partition $\mathcal{P}_k$ which evenly partitions $V$ by weight;
2 for $2 \le i \le f$ do
3     for each keyword $w$ between $l(c_{i-1})$ and $r(c_i)$ do
4         Compute $C(\mathcal{P}_k)$ suppose $c_{i-1}$ and $c_i$ are separated by
          $w$;
5         Update $c_{i-1}$ and $c_i$ in $\mathcal{P}_k$ using $w$ if a lower $C(\mathcal{P}_k)$ is
          achieved;
6 return $\mathcal{P}_k$
```

(2) Heuristic Partition.



(a) Example of GBD problem



(b) Example of our problem

Fig. 8. Example of NP-complete

Following the *local improvement heuristic* [29], we develop an efficient greedy partition algorithm, where details are illustrated in Algorithm 3. Line 1 first partitions V into f cuts with similar weights. Then Lines 2-5 iteratively improve keyword partition method by exhaustive search in a local area. In particular, let c_i denote the i-th ordered cut, while $l(c_i)$ and $r(c_i)$ represent its left and right boundary keywords, respectively. For each cut c_i $(1 < i \leq f)$, we attempt to reduce the *local* cost (i.e., the cost of c_{i-1} and c_i) by exhausting all possible boundary (separate) keywords regarding two adjacent cuts c_{i-1} and c_i. The time cost of Algorithm 3 is $O(f \times |V|)$ in the worst case.

C. Spatial Partition

Without loss of generality, we assume $f = m \times n$ and $\mathcal{P}_s$ represents a spatial partition of the node N which divides the region into $m \times n$ grid cells (buckets). We first show that it is an NP-hard problem to find optimal spatial partition. Then we resort to local improvement heuristic algorithm.

Theorem 1. *The problem of finding optimal spatial partition is **NP-hard**.*

Proof. Our proof relies on the problem of Generalized Block Distribution (GBD) [30] with $K = 1$, which is NP-complete.

GBD Instance: Given a $g \times g$ matrix $\mathbf{A}$, and each element is an integer; A partition which divides $\mathbf{A}$ into $m \times n$ contiguous blocks where $\mathbf{B}_{i,j}$ denotes the ij-th block; A function ϕ, where $\phi(\mathbf{B}_{i,j})$ reports the number of non-zero elements in block $\mathbf{B}_{i,j}$.

Question: Is there a partition on $\mathbf{A}$ such that

$$\max_{1 \leq i \leq m, 1 \leq j \leq n} \phi(\mathbf{B}_{i,j}) \leq 1 \tag{5}$$

Fig. 8(a) shows an example of GBD problem where each block contains at most one non-zero element under the given partition ($g = 4$, $m = n = 3$). Given an instance of GBD, we reduce it to a special case of decision version of our spatial partition problem as follows. Suppose there are $g \times g$ unit cells in the region of node N, as shown in Fig. 8(b) we put a query with extremely small region (thus being regarded as a point) at the center of an unit cell if the corresponding element in $\mathbf{A}$ is non-zero. A spatial partition of node N divides the space into $m \times n$ grid cells (buckets). $\mathbf{w}(B_{i,j})$ $(1 \leq i \leq m$, $1 \leq j \leq n)$ is the number of queries in the bucket $B_{i,j}$ and $\mathbf{p}(B_{i,j}) = \frac{\mathbf{w}(B_{i,j})}{|Q|}$ where $|Q|$ is the number of queries generated. A special case of decision version of our problem is that if there is a spatial partition $\mathcal{P}_s$ on the node N such that

$$C(\mathcal{P}_s) = \sum_{i=1}^{m}\sum_{j=1}^{n} \mathbf{w}(B_{i,j}) \times \mathbf{p}(B_{i,j}) \leq 1 \tag{6}$$

Since $\mathbf{p}(B_{i,j}) = \frac{\mathbf{w}(B_{i,j})}{|Q|}$, we have $C(\mathcal{P}_s) = \frac{1}{|Q|}\sum_{i=1}^{m}\sum_{j=1}^{n} \mathbf{w}(B_{i,j})^2$. Given the fact that $\sum_{i=1}^{m}\sum_{j=1}^{n} \mathbf{w}(B_{i,j}) = |Q|$, a partition $\mathcal{P}_s$ with $C(\mathcal{P}_s) \leq 1$ implies that $\mathbf{w}(B_{i,j}) \leq 1$ for any bucket $B_{i,j}$, i.e., there is at most one query in each bucket. Note that, if there exists one delimiter line of the spatial partition which lies across unit cells, we can simply shift it to its nearest boundary line without changing the partition cost. Consequently, as illustrated in Fig. 8, $\mathcal{P}_s$ immediately leads to a solution of the GBD problem in which there is at most one non-zero element in each block, and vice versa. Thus, our problem is NP-hard. □

Due to the NP-hardness of the problem, we resort to a *local improvement* heuristic algorithm in which the space is partitioned along each dimension independently. We first partition the space into m buckets along the first dimension such that the centers of the queries are evenly distributed. With similar rationale to Algorithm 3, we iteratively improve the partition cost. Since the possible number of boundary points along each dimension is bounded by $2 \times |Q|$, the time complexity is $O(m \times |Q|)$ in the worst case. Similarly, the space is partitioned into n buckets along another dimension. In this way, we divide the region of N into f grid cells with time complexity $O(\sqrt{f} \times |Q|)$.

D. Index Construction

Algorithm 4 presents the procedure of AP-Tree construction, which recursively divides queries through keyword and spatial partitions. Given a set Q of queries passed from parent node, the current node N may be set to *q*-node, *k*-node or *s*-node. Specifically, two flags, kP and sP, are used to indicate if queries in Q can be further partitioned by keyword and space, respectively. Line 2 keeps all queries in a *q*-node if the number of queries does not exceed a given threshold θ_q (i.e., $|Q| < \theta_q$) or queries cannot be split further by keyword or spatial partitions (i.e., kP is $false$ and sP is $false$). If keyword partition is allowed (i.e., kP is $true$), Line 6 explores keyword partition with offset l, and the cost is recorded by C_k. Recall that offset l indicates that the l-th keywords from queries in Q are employed for *keyword partition*. By C_s we record the cost of spatial partition at Line 8 if sP is $true$. Then we can decide the current node N to be constructed from keyword partition (Line 10) or spatial partition (Line 18) based on C_k and C_s. The queries in Q are pushed to related child nodes (i.e., cuts and cells) for further processing (Line 16 and Line 24), in which the partition offset is increased by one if keyword partition is adopted.

In addition to regular cuts (cells), we also maintain dummy cut (cell) for *k*-node (*s*-node). In particular, we maintain a dummy cut for a *k*-node such that queries whose keywords have been exhausted (i.e., $|q.\psi| < l$) are pushed to the dummy cut with kP set to $false$ (Lines 11-13). Similarly, Lines 19-21 push all queries with regions containing the node N to the dummy cell for further potential keyword partition, where the flag sP is set to $false$. Finally, the AP-Tree can be constructed by the function **BuildIndex**$(root, Q, 1, true, true)$.

Algorithm 4: **BuildIndex**(N, Q, l, kP, sP)

```
Input  : $N$ : current node, $Q$ : a set of queries
         $l$ : keyword partition offset to be used in $N$
         $kP$ and $sP$ : flags for keyword and spatial partitions
Output : AP-Tree
1  if ($kP$ is $false$ and $sP$ is $false$) or $|Q| < \theta_q$ then
2      $N$ is a $q$-node for $Q$;
3      return
4  $C_k := +\infty$; $C_s := +\infty$;
5  if $kP$ is $true$ then        /* Try keyword partition */
6      $C_k \leftarrow$ keyword partition on $Q$ with offset $l$;
7  if $sP$ is $true$ then        /* Try spatial partition */
8      $C_s \leftarrow$ spatial partition on $Q$;
9  if keyword partition is chosen (i.e., $C_k < C_s$) then
10     $N$ is a $k$-node with node offset $N_l = l$;
11     $Q' \leftarrow$ queries $\{q\}$ in $Q$ with $|q.\psi| < l$ ;
12     $B' \leftarrow$ dummy cut of $N$;
13     BuildIndex( $B'$, $Q'$, $l+1$, $kP = false$, $sP$) ;
14     for each child node (i.e., cut) $B$ of node $N$ do
15         $Q_B \leftarrow$ queries in $Q - Q'$ which hit the cut $B$ ;
16         BuildIndex($B$, $Q_B$, $l+1$, $kP$, $sP$);
17 else
18     $N$ is a $s$-node;
19     $Q' \leftarrow$ queries in $Q$ which contains $N_r$;
20     $B' \leftarrow$ dummy cell of $N$;
21     BuildIndex($B'$, $Q'$, $l$, $kP$, $sP = false$) ;
22     for each child node (i.e., cell) $B$ of node $N$ do
23         $Q_B \leftarrow$ queries in $Q - Q'$ which overlap or contain $B$ ;
24         BuildIndex( $B$, $Q_B$, $l$, $kP$, $sP$) ;
```

E. Index Maintenance

In practice, we may need to dynamically maintain an AP-Tree due to registration of new queries and deregistration of existing queries. A simple strategy is that we put a new query into its corresponding query node based on its ordered query keywords and query region, and a query node is partitioned when its number of queries exceeds the threshold θ_q. Similarly, we remove a query from its corresponding query nodes if it is deregistered and a keyword node or spatial node turns to a query node if the number of its descendant queries is less than θ_q. This approach is efficient and works well if the underlying query workload remains stable. On the downside, the partitions of the existing nodes cannot be adjusted to the change of query workload, and hence the performance may be deteriorated. To alleviate this issue, we adopt the well-known *KL-Divergence* [31] to detect the changes of underlying query workload for nodes with a particular amount of queries. Specifically, let $\mathbf{w}_{old}(B_i)$ denote the weight of the bucket B_i when the node is constructed while $\mathbf{w}(B_i)$ is calculated for all current queries. Let $D_{KL}(\mathbf{w}_{old}|\mathbf{w})$ denote *KL-Divergence* of the query workload, and an AP-Tree node will be re-constructed if $D_{KL}(\mathbf{w}_{old}|\mathbf{w})$ exceeds a given threshold θ_{KL}. We remark that calculation of *KL-Divergence* value is almost cost-free because they can be easily updated when the node is visited during the query updates. Moreover, only descendant queries of the node are involved in the re-construction. In this way, our empirical study shows that AP-Tree is self-adjustable to the workload changes with a decent maintenance overhead.

V. Experiment

In this section, we present the results of a comprehensive performance study to evaluate the effectiveness and efficiency of our techniques proposed in this paper.

A. Experiment Setup

To the best of our knowledge, IQ-Tree [4] and R^t-Tree [5] are only two existing work investigating continuous queries on streaming spatial-textual data. Both work fall in the category of *spatial-first* indexing structure. For comprehensive performance evaluation, we also investigate a *keyword-first* indexing structure, namely RQ-Tree. In this paper, we implement and evaluate following algorithms.

- **R^t-Tree.** Object matching algorithm based on R^{t++}-Tree proposed in [5], which achieves the best performance compared with R^t-Tree and R^{t+}-Tree. The source code is provided by the authors in [5].
- **IQ-Tree.** Object matching algorithm based on IQ-Tree proposed in [4]. The query decomposition cost model[4] is adopted to allocate queries to Quadtree cells according to query and object workloads.
- **RQ-Tree.** The representative of *keyword-first* indexing method which can be regarded as a variant of IQ-Tree. Particularly, RQ-Tree first employ ranked-key Inverted List [25], [4] to partition queries into the posting lists according to their *least frequent* keywords. Then for queries on each posting list, we build a Quadtree for spatial filtering purpose where the cost model in [4] is also adopted.
- **AP-Tree.** AP-Tree based object matching algorithm proposed in this paper. By default, the heuristic algorithms are employed for keyword and spatial partitions.

Datasets. Four datasets are collected for experimental evaluations. **TWEETS** is a real-life dataset collected from Twitter [5], containing 12 million tweets with geo-locations from May 2012 to August 2012. *TWEETS* is the default dataset in the experiments. **GN** is obtained from the US Board on Geographic Names[5] in which each object is associated with a geo-location and a short text description. **CARS** and **AIS** obtain the geo-locations from Chorochronos Archive[6] and we randomly tag the locations with user-generated keywords from 20 Newsgroups [7]. The statistics of four datasets are summarized in Table II.

Query Workload. We generate four query workloads based on the above four datasets. In each query workload, $5M$

[4]As we assume indexes are fit in the main memory, we use the number of verifications to evaluate the goodness of the query decomposition, instead of the number of I/Os.

[5]http://geonames.usgs.gov

[6]http://www.chorochronos.org

[7]http://people.csail.mit.edu/jrennie/20Newsgroups

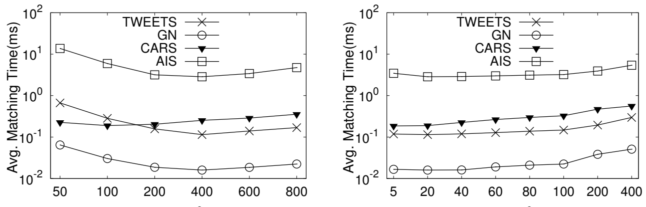
TABLE II
DATASETS STATISTICS

Datasets	TWEETS	GN	CARS	AIS
# objects	12.7M	2.2M	2.2M	5.7M
vocabulary size	1.7M	208K	81K	81K
avg. # keywords in objects	9	7	30	50

spatial-textual objects are randomly chosen from corresponding dataset. For each sampled object, we randomly pick m terms as query keywords and m is a random number between 1 and 5. The query region is set to a rectangle centered at the geo-location of the object, and the region size is uniformly chosen between 0.01% and 1% of the data space.

Object Workload. We use first 5% of the spatial-textual objects as the historical object workload when IQ-Tree, RQ-Tree and AP-Tree are constructed. The remaining objects are fed to the continuous queries as streaming spatial-textual data.

All experiments are implemented in C++. The experiments are conducted on a PC with 3.4GHz Intel Xeon 2 cores CPU and 32GB memory running Red Hat Enterprise Linux. Following the typical setting of publish/subscribe systems (e.g., [5], [17]), we assume indexes are fit in the main memory to support real-time response. The *average* object matching time is reported to evaluate the performance of the algorithms. We also evaluate the index construction and maintenance time as well as the index size. By default, keywords are ordered by their term frequencies over the query keywords (i.e., *Tf* order). Important parameters of AP-Tree and alternative implementations are investigated in Section V-B. Throughout the experiments, we set fanout f, partition threshold θ_q and *KL-Divergence* threshold θ_{KL} to 200, 40 and 0.001 respectively, unless otherwise specified.



(a) Vary f (b) Vary θ_q

Fig. 9. Effect of varying f and θ_q

B. Experimental Tuning

Effect of f and θ_q. In the first set of experiments, we evaluate the impact of the fanout f and partition threshold θ_q in four datasets under default settings. Intuitively, a small f cannot fully utilize the keyword partition due to the small number of cuts on each keyword node. On the other hand, a large f may result in poor adaptiveness of the AP-Tree. This is confirmed in Fig. 9(a), where the average matching time is reported with f varying from 50 to 800. We set f to 200 for all datasets in the hereafter experiments. Fig. 9(b) reports the average matching time as a function of θ_q which grows from 5 to 400. It is observed that θ_q does not noticeably affect performance when θ_q is smaller than 40. By default, θ_q is set to 40 for a better trade-off between index size and matching performance.

Effect of different order strategies. Fig. 10 evaluates the effect of three typical keyword order strategies on four datasets.

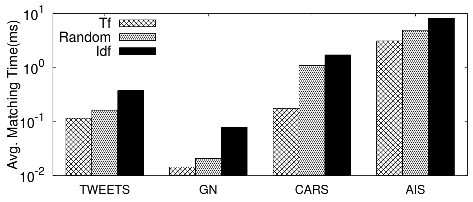
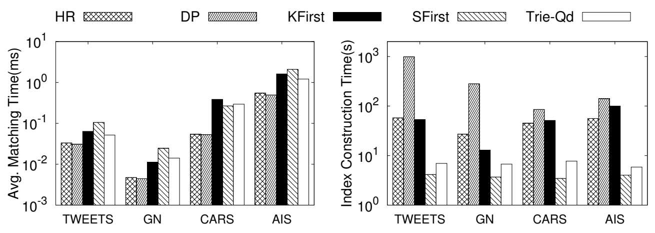


Fig. 10. Effect of keyword order



(a) Avg. Matching Time (b) Index Construction Time

Fig. 11. Comparison among different AP-Tree variants

In particular, keywords are sorted decreasingly according to their term frequencies in **Tf**, while **Idf** sorts keywords by the inverse document frequency, where a rare keyword is ranked high. Keywords are randomly ordered when **Random** is adopted. Fig. 10 demonstrates the superior performance of *Tf* order strategy on all datasets. The reason is that, given the fixed number of cuts on each keyword node, there are less number of false positives if the less frequent keywords are pushed to the lower level (i.e., nodes with larger offset values). *Tf* order is used in hereafter experiments.

Comparison of AP-Tree variants. We compare the performance of several variants of AP-Tree as follows. **DP** employs dynamic programming approach to find optimal *keyword partition*, and **HR** uses the heuristic keyword partition. **KFirst** puts high priority to keyword partition on each node when AP-Tree is constructed, while spatial partition is prioritized in **SFirst**. Finally, **Trie-Qd** adopts the ordered keyword trie structure in [7] to organize queries, and then uses Quadtree to further partition queries with the same query keywords. Fig. 11(a) and Fig. 11(b) report the average object matching cost and the index construction time of the algorithms, respectively, over four datasets where the default average query region size is set to 0.001%. Following are two important observations.

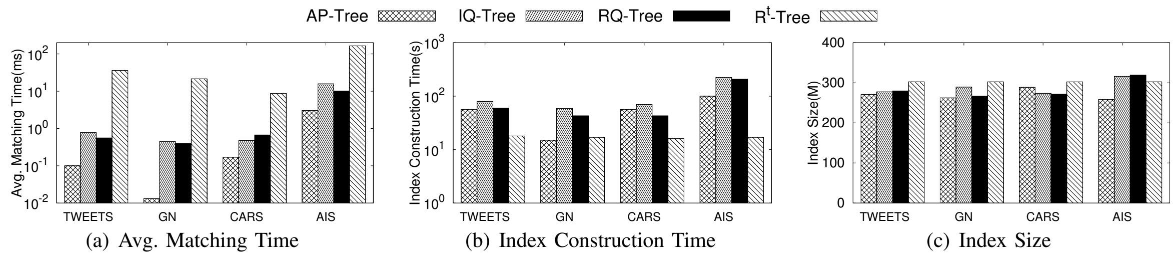
- Among all algorithms, *DP* achieves the best matching performance. *HR* has similar matching time with *DP* but beats *DP* by a huge margin w.r.t. index construction time.
- The poor matching performance of *KFirst* and *SFirst* implies that AP-Tree should be constructed in an adaptive way. Similarly, due to the lack of the adaptiveness and a large number of tree nodes, a straightforward combination of the ordered keyword trie [7] and Quadtree (*Trie-Qd*) cannot well support continuous spatial-keyword queries.

In hereafter experiments, *HR* is employed for performance evaluation of AP-Tree.

C. Performance Evaluation

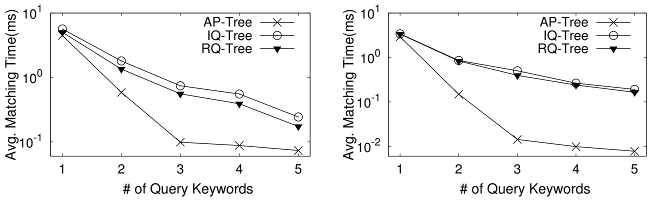
In this subsection, we evaluate the performance of AP-Tree with other competitors.

Evaluation on different datasets. We evaluate the average object matching time, index construction time and index size



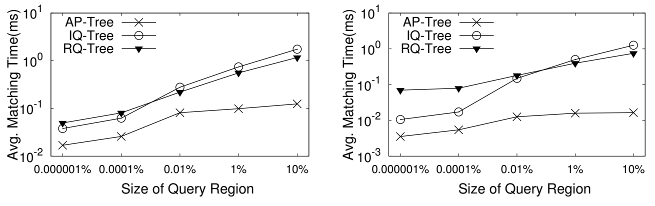
(a) Avg. Matching Time (b) Index Construction Time (c) Index Size

Fig. 12. Performance over various datasets



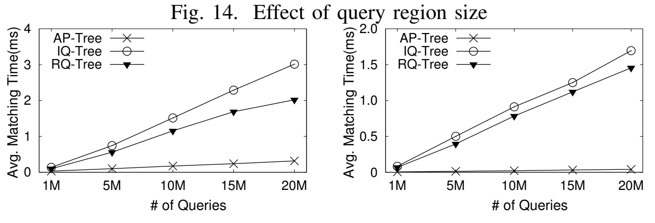
(a) TWEETS (b) GN

Fig. 13. Effect of number of query keywords



(a) TWEETS (b) GN

Fig. 14. Effect of query region size



(a) TWEETS (b) GN

Fig. 15. Effect of scalability

of the algorithms against four datasets *TWEETS*, *GN*, *CARS* and *AIS*. As shown in Fig. 12(a), AP-Tree significantly beats other algorithms in terms of object matching time. Particularly, AP-Tree is 30 times faster than the second best algorithm in *GN* because it is observed that the keyword and spatial distributions vary significantly among different regions in *GN*, and AP-Tree can take great advantage of its adaptiveness. It is worth noting that the *keyword-first* method RQ-Tree has better performance than two *spatial-first* methods (i.e., IQ-Tree and R^t-Tree) on *TWEETS*, *GN* and *AIS* datasets, but is defeated on *CARS* dataset by IQ-Tree. This implies that the effectiveness of the keyword and spatial filtering depends on the underlying query workload. As expected, Fig. 12(b) reports that R^t-Tree has the fastest index construction time because there is no cost model in [5] and the query regions are not decomposed. Fig. 12(c) shows that four index structures have similar index sizes. In the following experiments, we exclude R^t-Tree from the performance evaluation because it is dominated by IQ-Tree. Moreover both algorithms belong to *spatial-first* category, and hence exhibit similar trend in the experiments.

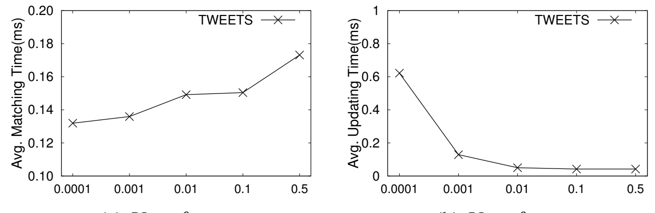
Effect of the number of query keywords. Fig. 13 evaluates the performance of three algorithms against *TWEETS* and *GN* datasets where the number of query keywords varies from 1 to 5. Not surprisingly, the performance of three algorithms improves with the growth of the number of query keywords because the number of matched queries is significantly reduced. When there is only one query keyword, AP-Tree only slightly outperforms RQ-Tree and IQ-Tree because it is difficult to distinguish queries from keyword perspective. Nevertheless, the margin becomes significant when there are more than one query keyword.

Effect of the query region size. We evaluate the effect of query region size in Fig. 14 where the average matching time is reported as a function of the region size varying from 0.000001% to 10% of the data space. As expected, the performance of three algorithms is sensitive to the region size because larger region size increases the number of matched queries and hence leads to higher matching costs. It is noticed that RQ-Tree is ranked after IQ-Tree when the region size is very small while RQ-Tree has better performance when the region size becomes large. This is quite intuitive because *spatial-first* is more attractive when the region size is very small. AP-Tree is the most stable algorithm and consistently beats RQ-Tree and IQ-Tree by a large margin. It is observed that more keyword nodes appear on high levels of AP-Tree when the query region is large, which verifies the adaptiveness of AP-Tree structure.

Effect of the number of queries. We turn to evaluate the scalability of the algorithms in Fig. 15 where the number of queries grows from $1M$ to $20M$. The result shows that AP-Tree is much more scalable to the number of queries. For instance, it only takes 0.4ms and 0.04ms on average to match incoming objects on *TWEETS* and *GN* datasets when the number of queries reaches 20M.

Evaluate index maintenance. We evaluate the costs of incremental maintenance of AP-Tree, IQ-Tree and RQ-Tree as well as their object matching performance. In particular, *TWEETS* dataset is deployed because the arrival order of the queries can naturally follow the corresponding timestamps of the tweets. The first δ percentage of the queries are used to construct the indexes and then remaining queries are incrementally inserted, where δ is set to 20 by default. Finally, we report the average object matching cost after all queries arrives. We also record the average updating time for all queries inserted.

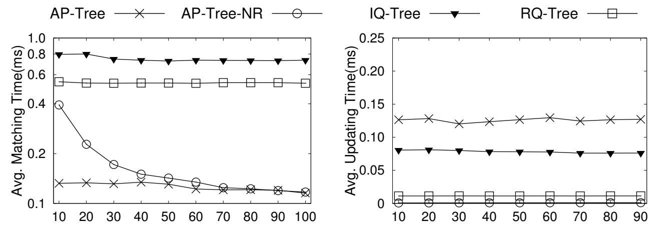
In the experiments, a keyword or spatial node of AP-Tree is re-constructed if it covers at least 0.1% of the query population and its *KL-divergence* value exceeds θ_{KL}. It is quite intuitive that a small θ_{KL} value results in a better object matching time but higher AP-Tree maintenance overhead. Fig. 16 evaluates



(a) Vary θ_{KL} (b) Vary θ_{KL}

Fig. 16. Effect of θ_{KL}

the impact of threshold θ_{KL} which increases from 0.0001 to 0.5. In the following experiments, we set θ_{KL} to 0.001 since it achieves a good trade-off between matching cost and maintenance cost.



(a) Vary δ (b) Vary δ

Fig. 17. Performance of maintenance

In the last set of experiments, we also consider a variant of AP-Tree algorithm, namely AP-Tree-NR, which does not re-construct the existing AP-Tree node. Fig. 17 reports the average object matching time as well as the average delay of query insertions for four algorithms where the percentage of queries used for initial AP-Tree construction ($\delta\%$) increases from 10% to 90%. Fig. 17(a) shows that the performance of AP-Tree-NR is not satisfactory when δ is small. This is because AP-Tree structure built on a small proportion of the query set does not well-suit to the change of query workload. On the contrary, the performance of AP-Tree is rather stable and consistently beats IQ-Tree and RQ-Tree by a large margin since AP-Tree can adjust the tree structure to the change of query workload by node re-constructions. The average maintenance cost of four algorithms is reported in Fig. 17(b). As expected, AP-Tree-NR has the best performance since there is no node re-constructions, while AP-Tree has the largest index maintenance overhead. Nevertheless, AP-Tree can process a query in around 0.12ms on average which is still quite efficient in practice.

VI. Conclusion

The phenomenon of streaming spatial-textual data raises interesting challenges for indexing continuous spatial-keyword queries. In this paper, we propose a novel adaptive spatial-textual partition indexing structure, namely AP-Tree, to efficiently organize a massive number of continuous spatial-keyword queries such that each incoming object from spatial-textual data can be rapidly delivered to relevant queries. Unlike the previous spatial-textual indexes which prefer either textual feature or spatial feature, AP-Tree can be constructed in an adaptive way by carefully choosing keyword or spatial partitions guided by a cost model. Extensive experiments demonstrate that our technique achieves a high throughput performance over streaming spatial-textual data.

Acknowledgment

Ying Zhang is supported by ARC DE140100679 and DP130103245. Wenjie Zhang is supported by ARC DP150103071, DP150102728 and DE120102144. Xuemin Lin is supported by ARC DP150102728, DP140103578 and NSFC61232006. Wei Wang is supported by ARC DP130103401 and DP130103405.

References

[1] T. W. Yan and H. García-Molina, "Index structures for selective dissemination of information under the boolean model," *TODS*, 1994.
[2] M.-H. Park, J.-H. Hong, and S.-B. Cho, "Location-based recommendation system using bayesian users preference model in mobile devices," in *Ubiquitous Intelligence and Computing*. Springer, 2007.
[3] A. Konig, K. Church, and M. Markov, "A data structure for sponsored search," in *ICDE*, 2009, pp. 90–101.
[4] L. Chen, G. Cong, and X. Cao, "An efficient query indexing mechanism for filtering geo-textual data," in *SIGMOD*, 2013, pp. 749–760.
[5] G. Li, Y. Wang, T. Wang, and J. Feng, "Location-aware publish/subscribe," in *ACM SIGKDD*, 2013, pp. 802–810.
[6] S. Helmer and G. Moerkotte, "A performance study of four index structures for set-valued attributes of low cardinality," *VLDBJ*, 2003.
[7] Z. Hmedeh, H. Kourdounakis, V. Christophides, C. Du Mouza, M. Scholl, and N. Travers, "Subscription indexes for web syndication systems," in *EDBT*, 2012, pp. 312–323.
[8] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel, "Text vs. space: efficient geo-search query processing," in *CIKM*, 2011.
[9] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-k spatial keyword queries," in *SSTD*, 2011.
[10] C. Zhang, Y. Zhang, W. Zhang, and X. Lin, "Inverted linear quadtree: Efficient top k spatial keyword search." in *ICDE*, 2013, pp. 901–912.
[11] C. Zhang, Y. Zhang, W. Zhang, X. Lin, M. A. Cheema, and X. Wang, "Diversified spatial keyword search on road networks," in *EDBT*, 2014.
[12] I. De Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *ICDE*, 2008, pp. 656–665.
[13] 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.
[14] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha, "Filtering algorithms and implementation for very fast publish/subscribe," in *SIGMOD Conference*, 2001, pp. 115–126.
[15] S. E. Whang, H. Garcia-Molina, C. Brower, J. Shanmugasundaram, S. Vassilvitskii, E. Vee, and R. Yerneni, "Indexing boolean expressions," *PVLDB*, vol. 2, no. 1, pp. 37–48, 2009.
[16] M. Sadoghi and H.-A. Jacobsen, "Be-tree: An index structure to efficiently match boolean expressions over high-dimensional discrete space," in *ACM SIGMOD*, 2011, pp. 637–648.
[17] D. Zhang, C.-Y. Chan, and K.-L. Tan, "An efficient publish/subscribe index for e-commerce databases," *PVLDB*, vol. 7, no. 8, 2014.
[18] K. Mouratidis and H. Pang, "Efficient evaluation of continuous text search queries," *IEEE TKDE*, vol. 23, no. 10, pp. 1469–1482, 2011.
[19] A. Shraer, M. Gurevich, M. Fontoura, and V. Josifovski, "Top-k publish-subscribe for social annotation of news," *PVLDB*, 2013.
[20] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, "Efficient continuously moving top-k spatial keyword query processing," in *ICDE*, 2011.
[21] W. Huang, G. Li, K.-L. Tan, and J. Feng, "Efficient safe-region construction for moving top-k spatial keyword queries," in *CIKM*, 2012.
[22] X. Chen, Y. Chen, and F. Rao, "An efficient spatial publish/subscribe system for intelligent location-based services," in *DEBS*, 2003.
[23] J. Bao, M. F. Mokbel, and C.-Y. Chow, "Geofeed: A location aware news feed system," in *ICDE*, 2012, pp. 54–65.
[24] L. Chen, G. Cong, X. Cao, and K.-L. Tan, "Temporal spatial-keyword top-k publish/subscribe," in *ICDE*, 2015.
[25] T. W. Yan and H. Garcia-Molina, "Duplicate removal in information system dissemination," in *PVLDB*, 1995, pp. 66–77.
[26] M. Terrovitis, P. Bouros, P. Vassiliadis, T. K. Sellis, and N. Mamoulis, "Efficient answering of set containment queries for skewed item distributions," in *EDBT*, 2011, pp. 225–236.
[27] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf, *Computational geometry*. Springer, 2000.
[28] J. L. Bentley, "Solutions to klees rectangle problems," Technical report, Carnegie-Mellon Univ., Pittsburgh, PA, Tech. Rep., 1977.
[29] A. N. Swami, "Optimization of large join queries: Combining heuristic and combinatorial techniques," in *SIGMOD*, 1989, pp. 367–376.
[30] M. Grigni and F. Manne, "On the complexity of the generalized block distribution," in *Parallel Algorithms for Irregularly Structured Problems*. Springer, 1996, pp. 319–326.
[31] S. Kullback, *Information theory and statistics*. Courier Dover Publications, 1997.