Efficient Top-k Subscription Matching for Location-Aware Publish/Subscribe

Jiafeng Hu[1(✉)], Reynold Cheng[1], Dingming Wu[1], and Beihong Jin[2]

[1] Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong, China
{jhu,ckcheng,dmwu}@cs.hku.hk

[2] State Key Laboratory of Computer Science,Institute of Software, Chinese Academy of Sciences, Beijing, China
beihong@iscas.ac.cn

Abstract. The dissemination of messages to a vast number of mobile users has raised a lot of attention. This issue is inherent in emerging applications, such as location-based targeted advertising, selective information disseminating, and ride sharing. In this paper, we examine how to support location-based message dissemination in an effective and efficient manner. Our main idea is to develop a location-aware version of the *Pub/Sub* model, which was designed for message dissemination. While a lot of studies have successfully used this model to match the interest of *subscriptions* (e.g., the properties of potential customers) and *events* (e.g., information of casual users), the issues of incorporating the location information of subscribers and publishers have not been well addressed. We propose to model subscriptions and events by boolean expressions and location data. This allows complex information to be specified. However, since the number of publishers and subscribers can be enormous, the time cost for matching subscriptions and events can be prohibitive. To address this problem, we have developed the R^I-tree. This data structure is an integration of the R-tree and the dynamic interval-tree. Together with our novel pruning strategy on R^I-tree, our solution can effectively and efficiently return the top-k subscriptions with respect to an event. We have performed extensive evaluations to verify our approach.

1 Introduction

Due to the advance of telecommunications and Internet technologies, tremendous amounts of location information can now be obtained easily. For instance, a user's location is often tracked by base stations in a cellular network; a vehicle's position can be obtained through GPS receivers or sensors on roads; a user reveals her location when she "checks in" (e.g., through Facebook and Twitter). The availability of location information stimulates the development of location-based messaging services, which disseminates interesting messages to users based on their positions and other information. Taking location-based targeted advertising as an example, advertisements are sent to users selected in terms of age,

C. Claramunt et al. (Eds.): SSTD 2015, LNCS 9239, pp. 1–19, 2015.
DOI: 10.1007/978-3-319-22363-6_18

gender, interest, and location[1]. Another example is the Location-Based App Recommendation (LBAR), where software or "apps" are suggested to a user based on where she is. An LBAR feature recently appears in Apple's iOS 8[2], which shows the picture of an app (e.g., "Starbucks") in the lock screen based on the user's context (e.g., she is close to a Starbucks coffee shop). As pointed out by Verve Mobile in 2013, the LBTA outperforms non-location-targeted advertising by a factor of two, and the usage of the LBTA exceeds the industry average click-through rate (CTR) of 0.4 %[3].

Those applications can be built on the top of Publish/Subscribe (Pub/Sub) systems [6] which can provide large-scale matching and information dissemination. In a Pub/Sub system, there are two kinds of clients, *subscriber* and *publisher*. A subscriber, typically an information provider such as an advertising company, specifies the properties of users in which it is interested. These properties, or constraints, are collectively known as a subscription. For instance, an advertising company A (e.g., a restaurant), acting as a subscriber, posts the following subscription to the Pub/Sub system: $(15 < age < 30, interest = \{barbecue, sushi\}, gender = male, visited\_time \geq 3)$. The constraints specified in this subscription are used to match the "events" published by a publisher; once a matching is found, information from a subscriber is sent to the publisher. A publisher can be a casual mobile phone user. When a publisher, say, U, browses a homepage (say, Facebook), an event, containing information about this user (e.g., *age=25, interest=barbecue, gender=male, visited_time=5*), is sent to the system.

Since one or more constraints may be specified in a subscription, it may not be possible for the values of an event to match all the constraints. Hence, researchers have proposed to allow more flexibility in the matching process by allowing matching between subscribers and publishers to be inexact or partial. This variant of Pub/Sub systems, called ranked Pub/Sub systems [14,17,18], return the k best subscriptions (or top-k subscriptions) to a publisher based on some scoring functions.

We notice that the current work only focuses on normal boolean expressions including strings and numbers. However, the newly emerging applications bring the new technical challenges. Continuing the example of a location-based targeted advertising application, when a publisher opens an app, his geographic coordinates will be sent to the system. A subscriber can also specify this kind of location, for instance, by saying that his shop is at a specific location. On the other hand, to push advertisements to a mobile user, due to the factors like limited network bandwidth and the screen size of user's mobile phone, only the advertisements whose distribution scopes are near to the user's current location may become the candidate advertisements. Since subscriptions contain both complex boolean expressions and location information, it is rather costly to retrieve top-k relevant subscriptions from millions of subscriptions for an event.

[1] http://www.google.com/ads/admob/.
[2] http://goo.gl/wZeSg5.
[3] http://goo.gl/kJpQPG.

Different from the ranked Pub/Sub systems, the location information has been integrated into the so-called keyword Boolean matching Pub/Sub systems [11,13,19] where the numeric attribute matching is not supported. Their methods of incorporating the location information are either not applicable or inefficient for the ranked Pub/Sub systems. In the empirical studies, we extend the existing work related to the location-aware Pub/Sub systems as the competitors to support the numeric attribute matching. The experimental results demonstrate that our approach significantly outperforms the competitors.

In this paper, we explore the issues of incorporating location information into a ranked Pub/Sub system. To support top-k subscription matching for location-aware Pub/Sub systems, we propose a novel R-tree based index, the R^I-tree, by integrating the dynamic interval tree into the R-tree nodes. When an event with location information arrives, our algorithm can quickly report the top-k subscriptions most relevant to the event. To summarize, our main contributions are:

1. We formalize a new variant of top-k subscription matching, permitting location data to be a part of a subscription or an event;
2. We propose an index structure, called the R^I-tree;
3. We design an efficient matching algorithm; and
4. Our experimental evaluation validates the feasibility of our R^I-tree based solution.

The rest of the paper is organized as follows. Section 2 overviews the related work. Section 3 formulates the top-k subscription matching. Section 4 gives the threshold algorithm based solution as a baseline solution. In Sect. 5, we present the R^I-tree index and describe the matching algorithm. Finally, we evaluate the performance of the R^I-tree based solution by extensive experiments in Sect. 6 and conclude the paper in Sect. 7.

2 Related Work

The related work can be categorized into two main areas: ranked Pub/Sub systems and location-aware Pub/Sub systems. The differences between our R^I-tree solution and the existing solutions are summarized in Table 1 (BE denotes Boolean Expression).

Table 1. Comparison of existing location-aware Pub/Sub systems

	Pub/Sub					spatial keyword		R^I-tree
	SOPT-R-tree [14]	k-index [18]	BE*-tree [17]	R^t-tree [13]	OpIndex [20]	IR-tree [4]	I^3 [21]	
matching semantics	BE	BE	BE	keyword	BE	keyword	keyword	BE
location data	×	×	×	✓	×	✓	✓	✓
top-k	✓	✓	✓	×	×	✓	✓	✓

2.1 Ranked Pub/Sub Systems

Since the issue of the top-k subscription matching is posed in [14], there have existed some researches on top-k subscription matching [8,14,17,18]. [14] designates a subscription to be a set of intervals over a multi-dimensional space, in which each dimension is associated with a weight, and an event to be a point over the same multi-dimensional space. Based on this, [14] builds scored interval indexes for each dimension of the subscriptions. As thus, while an event arrives, the matching is carried out on every dimension of subscriptions, returning a corresponding subscription list sorted in a descending order of the scores. Then, the threshold algorithm [7] (TA for short) is employed to merge multiple sorted lists to obtain the subscriptions whose scores are ranked among the top k. Moreover, [14] presents two novel index structures: the IR-tree and the SOPT-R-tree to support top-k subscription matching. However, these index structures cannot support dynamic insertion and deletion of subscriptions. So they are not applicable to the scenarios where the subscriptions are updated frequently.

Whang et al. [18] turn the top-k subscription matching into another problem: how to efficiently index Disjunctive Normal Form (DNF) and Conjunctive Normal Form (CNF) boolean expressions over a high-dimensional space so as to quickly find the boolean expressions that evaluate to true for a given assignment of values to attributes. [18] presents the k-index, an inverted list based index for DNF and CNF boolean expressions, and then finds the top-k matched boolean expressions by virtue of the k-index. As an extension of [18], the methods presented in [8] are not restricted to the normal form expressions, but can deal with arbitrarily complex boolean expressions. [8] leverages existing techniques for evaluating leaf-level conjunctions, and then develops two bottom-up evaluation techniques, Dewey ID matching and Interval ID matching, to reduce unnecessary evaluation.

For the hierarchical top-k subscription matching, [17] presents a novel index structure named BE*-tree, which permits the values of attributes to be a continuous or discrete domain and combines a bi-directional tree expansion mechanism and an overlap-free splitting strategy to adapt to different workloads. For very high dimensional subscription matching, [20] proposes an in-memory index OpIndex, which builds an inverted index on the pivot attributes of subscriptions and designs a two-level partitioning scheme. However, OpIndex is not yet available to support location data and ranking.

2.2 Location-Aware Pub/Sub Systems

Recently, there have been many researches on location-aware Pub/Sub systems from a database perspective [2,9,11,13,19]. [13] proposes the R^t-tree which can efficiently filter geo-textual data. [19] extends R^t-tree to support ranking semantics, i.e., return all subscriptions whose similarities with the query event are not smaller than a given threshold θ. But the pruning algorithm proposed in [19] can not be used for top-k search. Further, [11] studies the location-aware Pub/Sub problem for parameterized spatio-textual subscriptions and presents

a filter-verification framework by integrating prefix filtering and spatial pruning techniques. However, only keywords are considered in [11,13,19], and they can not support to retrieve top-k subscription matching. Note that Pub/Sub systems which only consider keywords cannot support numeric attribute matching such as the example of the subscription described in Sect. 1. Compared to [11,13,19], the R^I-tree proposed in this paper has two distinguishing features. First, it allows users to specify their interests with boolean expressions, which is more expressive than keywords. Second, it focuses on the top-k semantics which is frequently used in many emerging applications (e.g., location-based targeted advertising).

Chen et al. [2] considers the temporal spatial-keyword top-k subscription query. They present an efficient solution which can continuously maintain up-to-date top-k most relevant results (events) over a stream of geo-textual objects for each subscription. [9] proposes a new location-aware Pub/Sub system, i.e., *Elaps*, that focuses on continuously monitoring moving users subscribing to dynamic event streams. However, their problems are different from ours. Our work is also different from spatial keyword search [1,4,21]. The main reason is that they focus on keywords while we adopt boolean expressions to express subscriber's requirements in subscriptions, which is more expressive than keywords.

3 Problem Formalization

3.1 Data Model

Definition 1. ***Subscription:*** *A subscription s contains a boolean expression Ω, a location loc, and a tuning parameter α, i.e., $s : \Omega \wedge loc \wedge \alpha$. The boolean expression is a conjunction of predicates, i.e., $\Omega = \{p_1 \wedge \cdots \wedge p_n\}$. A predicate is a quadruple, i.e., $p =< attr, op, val, \omega >$, with $attr$ being an attribute id that uniquely represents a dimension, op being an operator (e.g., from the relational operators $(<, \leq, =, \neq, \geq, >)$, the set operators $(\in_s, \notin_s)$ and the interval operator $(\in_i)$), val being a value, a set of values in discrete domains or a range of values in continuous domains, and ω being an assigned predicate weight, where $\sum_{i=1}^{n} p_i.\omega = 1$. The predicate weight signifies the relevance between the predicate and the event and can be given by subscribers (e.g., the advertiser assigns higher weights to more relevant predicates). The location loc represents the spatial dimension and is denoted as a conjunction of two triples, i.e., $(lat = val_{lat}) \wedge (lon = val_{lon})$, where lat (lon) denotes the latitude (longitude) of the object. The parameter α is used to balance the relative importance of non-spatial and spatial similarity.*

For example, in the targeted advertising, the subscription for an advertisement from a restaurant can be:$\{(age \in_i [15, 30], 0.3) \wedge (income > 5000, 0.4) \wedge (credit\_score > 80, 0.3) \wedge (lat = 22.27) \wedge (lon = 114.17) \wedge \alpha = 0.5\}$.

As explained in [16], predicates with different types of operators can be converted into one-dimensional intervals, as shown in Table 2 , where v_{min} and v_{max} are the smallest and the largest possible values in the corresponding domain, and

$\{v_1, \cdots, v_k\}$ is sorted in an ascending order. The way of converting $\neq$ and $\notin_s$ to an interval that spans the entire domain is built on the following reasonable speculation: these kinds of predicates are satisfied with a high probability by an event having a predicate on the corresponding attribute. Thus, the given transformation can help the early pruning during the matching. Therefore, in this paper, we focus on predicates in the form of intervals.

Table 2. Predicate Conversion

Predicates	Interval	Predicates	Interval
$i < v_1$	$[v_{min}, v_1)$	$i \geq v_1$	$[v_1, v_{max}]$
$i \leq v_1$	$[v_{min}, v_1]$	$i \in_s \{v_1, \cdots, v_k\}$	$[v_1, v_k]$
$i = v_1$	$[v_1, v_1]$	$i \notin_s \{v_1, \cdots, v_k\}$	$[v_{min}, v_{max}]$
$i \neq v_1$	$[v_{min}, v_{max}]$	$i \in_i [v_1, v_2]$	$[v_1, v_2]$
$i > v_1$	$(v_1, v_{max}]$		

Definition 2. ***Event:*** *An event e includes a non-spatial set of attribute name and value pairs and a location loc, i.e.,* $e : (attr_1 = val_1) \wedge (attr_2 = val_2) \wedge \cdots \wedge (attr_{|e|} = val_{|e|}) \wedge (lat = val_{lat}) \wedge (lon = val_{lon})$*, where* $attr_i$ *is the attribute identifier,* val_i *is the associated value,* val_{lat} *and* val_{lon} *are the publisher's current latitude and longitude, respectively.*

Here is an example of an event: $\{age = 25 \wedge income = 5000 \wedge credit\_score = 1000 \wedge lat = 22.27 \wedge lon = 114.17\}$.

Definition 3. ***Similarity Function*** ψ***:*** *Given event* $e : (attr_1 = val_1) \wedge (attr_2 = val_2) \wedge \cdots \wedge (attr_{|e|} = val_{|e|}) \wedge loc$ *and subscription* $s : (\Omega \wedge loc \wedge \alpha) = (p_1 \wedge p_2 \wedge \cdots \wedge p_n \wedge loc \wedge \alpha)$*, the similarity function* $\psi(e, s)$ *is defined as follows.*

$$\psi(e, s) = (1 - s.\alpha) \cdot \psi_t(e, s) + s.\alpha \cdot \psi_s(e, s), \tag{1}$$

where ψ_t *is a non-spatial similarity function and* ψ_s *is a spatial similarity function.*

Further, the non-spatial similarity[4] *is given by:*

$$\psi_t(e, s) = \sum_{e.attr_i \in e, s.p_j \in s.\Omega, e.attr_i = s.p_j.attr} s.p_j.\omega \cdot check(e.val_i, s.p_j) \tag{2}$$

where $e.val_i$ *denotes* i^{th} *attribute value of event e,* $s.p_j$ $(j = 1, \cdots, n)$ *denotes the* j^{th} *predicate of subscription s and function* $check(e.val_i, s.p_j)$ *is defined in Eq. 3 to check whether the constraint* $s.p_j$ *is satisfied by* $e.val_i$*.*

$$check(e.val_i, s.p_j) = \begin{cases} 1 & e.val_i \in_i s.p_j.val \\ 0 & otherwise \end{cases} \tag{3}$$

[4] More generally, the non-spatial similarity can be any monotonic function of the weights.

The spatial similarity is given by:

$$\psi_s(e, s) = 1 - \frac{dist(e.loc, s.loc)}{MaxDist}, \tag{4}$$

where $dist(e.loc, s.loc)$ *is the Euclidian distance between* $e.loc$ *and* $s.loc$*, and* $MaxDist$ *is the maximum Euclidian distance between subscriptions.*

3.2 Problem Definition

Based on the above definitions, we formulate the problem we will solve as follows. Given a set of subscriptions $\mathcal{S}$, an event e, and a parameter k, the *Top-k Subscription Matching problem* (SM-k problem for short) finds the top-k best matching set $\mathcal{S}_k \subseteq \mathcal{S}$ which is defined as $\mathcal{S}_k = \{s|\psi(e, s) \geq \psi(e, s'), \forall s' \in \mathcal{S} \backslash \mathcal{S}_k\}$ and $|\mathcal{S}_k| = k$.

Example 1. Figure 1 shows 9 subscriptions $s_0 \cdots s_8$ and an event e. The similarities between subscriptions and the event e are shown in Table 3. For event e, subscription s_0 is the result of the top-1 matching according to Eq. 1, i.e., $\psi(e, s_0) = \mathbf{0.925}$.

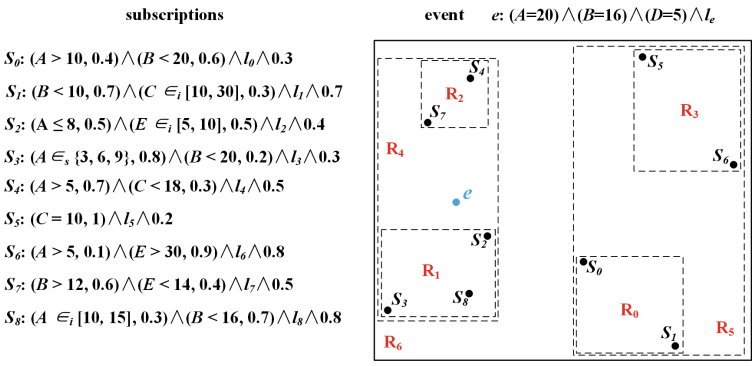


Fig. 1. Example of subscriptions and an event

Table 3. Similarities between event e and subscriptions in Fig. 1

s	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
$\psi_s(s, e)$	0.75	0.5	0.875	0.75	0.75	0.5	0.5	0.8	0.8
$\psi_t(s, e)$	1.0	0.0	0.0	0.2	0.7	0.0	0.1	0.6	0.0
$\psi(s, e)$	**0.925**	0.35	0.35	0.365	0.725	0.1	0.42	0.7	0.64

4 Baseline Solution

In this section, we present the *Threshold Algorithm-based Solution* for SM-k problem as a baseline solution. Considering that top-k subscription matching belongs to the relaxed matching and the subscription is not required to match with the event on each attribute exactly, we build an index for each attribute, i.e., build a scored segment-tree [14] for every non-spatial attribute and R-tree [10] for the spatial attribute. Based on that, we propose the *Threshold Algorithm based Solution (TAS)*.

TAS builds a two level index for subscriptions. At the root level, a hashmap is used to map attribute names to the sub-level data structures. At the sub-level, an index is built for each attribute to index those subscriptions contain that attribute.

For each non-spatial attribute, we build a scored segment-tree [14] for all subscription intervals in that attribute. The scored segment-tree is a variant of segment trees. A segment-tree [5] is a binary-tree structure to index intervals (segments). It partitions the intervals into a collection of disjoint, atomic intervals. Each atomic interval corresponds to a leaf node in the tree. If the length of the whole interval is n, then a segment tree is a balanced binary tree with n nodes as leaves and $\log n$ as the height of the tree.

Let $\mathcal{I}$ be the set of all subscription intervals in attribute $attr_i$, I be the interval constraint of subscription s in attribute $attr_i$ where $I \in \mathcal{I}$, $interval(V)$ denote the interval of node V, and $\mathcal{S}_V$ be the set of all subscriptions stored on node V, where $\forall I \in \mathcal{S}_V$, we have $interval(V) \subseteq I$ and $interval(U) \not\subseteq I$, here, node U is the father of node V.

To retrieve top-k scoring interval of $\mathcal{I}$ stabbed by an event point $e.val_i$, a segment-tree should be modified into a scored segment tree [14], i.e., subscriptions stored in node V are sorted in the order of their weights (in practice, it can be implemented by a priority queue). When retrieving top-k subscriptions in a scored segment-tree $\mathcal{T}$ for attribute $attr_i$, we maintain a global max-heap of size $O(\log n)$ and follow the steps below.

Firstly, we replenish the heap by inserting the top element of each subscription list from the nodes on the retrieval path in $\mathcal{T}$. Secondly, we pop out the top element of the heap as a candidate subscription (assuming that the element popped out is stored on node V) and insert the next element of the subscription list from node V. Run the second step in turns, until k elements are picked up. As thus, we can get top-k subscriptions (elements) on $\mathcal{T}$.

For the spatial attribute, we build an R-tree for all locations of subscriptions. On each node N of the R-tree, an extra information α_{max} is stored. If the node N is a leaf node, the value of α_{max} is the α of the corresponding subscription. Otherwise, if the node N is a non-leaf node, the value of α_{max} is the maximum of all its children's α_{max}. Thus, the new R-tree can support incremental nearest neighbor (NN) search. When an event location $e.loc$ arrives, for each non-leaf node N on the R-tree, the upper bound is

$$N.\alpha_{max} \cdot (1 - \frac{MinDist(e.loc, N.rectangle)}{MaxDist}),$$

where $MinDist(e.loc, N.rectangle)$ denotes the minimum Euclidian distance between $e.loc$ and any point on the *Minimum Bounding Box* of node N.

Now we describe the matching process. When an event e arrives, for each attribute in e, use the hashmap to get the related index structure, and incrementally return the subscriptions matching with e on that attribute in the order of their weight on that attribute (The weight of a subscription s on each attribute is multiplied by $(1 - s.\alpha)$ when s is inserted into the scored segment-tree). Then we use the Threshold Algorithm (TA) [7] to merge multiple ranked lists. It is proved that TA is correct and instance optimal in [7].

Thus, the steps of *TAS* are as follows:

1. At the beginning, for each attribute of the event, retrieve the best candidate subscription using the corresponding index structure. Then go to 3.
2. For each attribute of the event, retrieve the next best candidate subscription.
3. Merge those candidates using TA. If the terminal criterion in TA cannot be satisfied, then go to 2. Otherwise, go to 4.
4. Return the top-k subscriptions got in TA.

TAS will search on each attribute separately. If there exists a subscription matched with an event on many attributes with small weight for each predicate (the total similarity is very large), the searching list for each attribute can be very long. Thus, TAS is not very efficient. In order to avoid this problem, we design a novel index which combines all attributes on one tree index.

5 R^I-tree Based Solution

In this section, we present a framework that integrates the R-tree and the interval-tree into a new index, named R^I-tree and that includes an algorithm for processing SM-k problem using the R^I-tree.

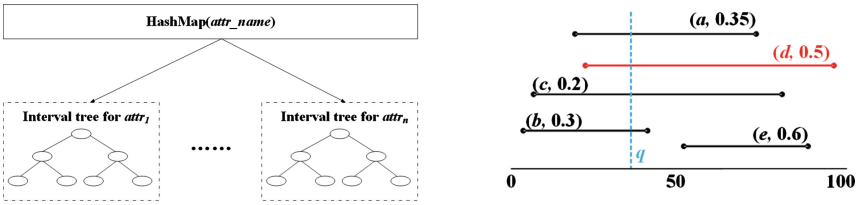
5.1 R^I-tree Index Structure

The R-tree [10] is a widely used index for spatial queries and the interval-tree [15] is the "standard" known solution for efficiently processing simple stabbing queries. They are designed separately for different kinds of queries.

The R^I-tree is essentially an R-tree, each node of which is enriched with reference to a set of dynamic interval trees for objects contained in its sub-tree.

In the R^I-tree, if node N is a leaf node, it contains a number of entries of the form $(sid, \Omega, loc, \alpha)$, where sid is the identifier of an subscription, Ω, loc and α are the boolean expression, the location and the tuning parameter of the subscription s_{sid}, respectively. Here, it is important to note that the weight of a subscription s_{sid} on each attribute is multiplied by $(1 - s_{sid}.\alpha)$ when s_{sid} is inserted into the R^I-tree. A leaf node also contains some metadata. The metadata includes $rectangle$, which is the Minimum Bounding Rectangle of all constituent entries, α_{min} and α_{max} which are the minimum and maximum value of α among all constituent entries, and the aggregated information Γ for each

attribute of the form $(attr_i, range, \omega_{max})$. In addition, a leaf node also contains a pointer to a dynamic interval tree forest $\mathcal{F}$, i.e., a set of dynamic interval trees organized by a hashmap, shown as Fig. 2a. Let $\mathcal{S}_{attr}$ be the set of all attributes stored on the leaf node. The hashmap manages all attributes in $\mathcal{S}_{attr}$. For each attribute $attr_i$, the hashmap maps it to a dynamic interval tree $\mathcal{T}_{attr_i}$. The tree $\mathcal{T}_{attr_i}$ stores all intervals of the leaf node N's entries on attribute $attr_i$. The form of intervals stored on the tree is $(range, sid, \omega)$, where $range$ denotes the range of the interval, sid is the id of the corresponding subscription and ω is the weight of that subscription on $attr_i$.



(a) dynamic interval tree forest (b) interval stabbing-max problem

Fig. 2. Example of the dynamic interval tree forest and the interval stabbing-max problem

The dynamic interval tree $\mathcal{T}_{attr_i}$ dynamically maintains a set of intervals $\mathcal{I}$, where each interval $I \in \mathcal{I}$ has a weight $I.\omega$ such that the interval with the maximum weight containing an event point can be found efficiently. This structure can solve the *interval stabbing-max* problem. For instance, as shown in Fig. 2b, for the query point q, it stabs four intervals (a, b, c, d). Since interval d has the greatest weight 0.5, it will be returned. There exists several solutions which can solve this problem. In this paper, we use the modified interval tree structure mentioned in [12] which can support queries in $O(\log^2 n)$ time, updates in $O(\log n)$ time and only requires $O(n)$ space.

On the other hand, if node N is a non-leaf node, it contains a number of entries cp, which points to the corresponding child node. Being same as the type of leaf nodes, node N also maintains the metadata and a dynamic interval tree forest organized by a two-level index structure which contains the interval information for each associated attributes. For each child node U of node N, the interval of $U.\Gamma_{attr_i}$ will be stored on the dynamic interval tree $\mathcal{T}_{attr_i}$ of the node N. Thus, the number of intervals in $\mathcal{T}_{attr_i}$ will not exceed the number of entries in node N.

Example 2. Figure 3 illustrates the R^I-tree index for the subscriptions in Fig. 1. Figure 4 is an example of $Metadata_1$ and $Forest_1$.

After describing the R^I-tree index, now we introduce an important metric, the *Upper Bound* (UB) of the similarity. Given an event e and a node N in the R^I-tree, the metric UB provides an upper bound of the similarity between the

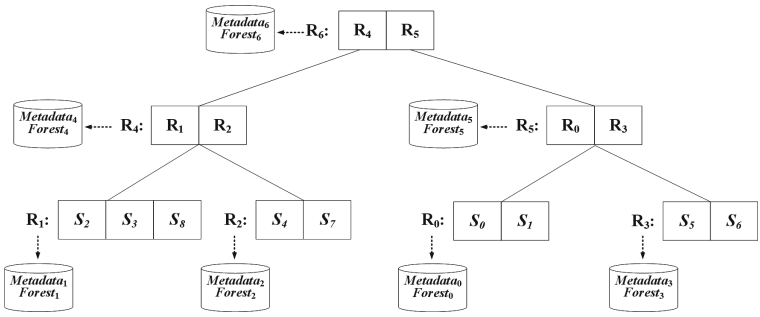


Fig. 3. R^I-tree index for subscriptions in Figure

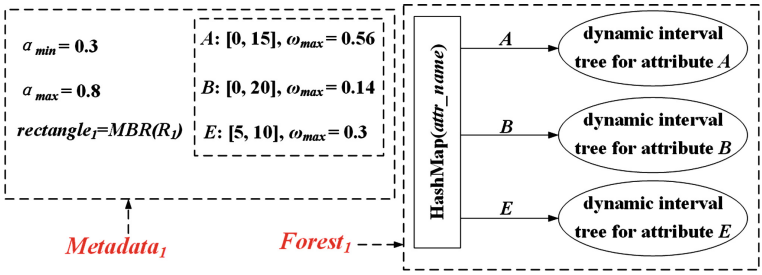


Fig. 4. Example of $Metadata_1$ and $Forest_1$

event e and all subscriptions located at the rectangle of node N. It can be used to order and efficiently prune the paths of the search space in the R^I-tree. To get the value of UB, we first calculate the upper bound of non-spatial similarity UB_t, and then calculate the upper bound of spatial similarity UB_s.

Definition 4. ***$UB_t(e, N)$:*** *Given an event e and a node N, the upper bound of non-spatial similarity $UB_t(e, N)$ is defined as follows:*

$$UB_t(e, N) = \sum_{e.attr_i = N.\Gamma.attr_j} N.\mathcal{T}_{attr_j}(e.val_i), \tag{5}$$

where $\mathcal{T}_{attr_j}(e.val_i)$ returns the maximum weight of the interval containing $e.val_i$ in attribute $N.\Gamma.attr_j$.

Definition 5. ***$UB_s(e, N)$:*** *Given an event e and a node N, the upper bound of spatial similarity $UB_s(e, N)$ is defined as follows:*

$$UB_s(e, N) = 1 - \frac{MinDist(e.loc, N.rectangle)}{MaxDist}, \tag{6}$$

where $MaxDist$ is the same as in Eq. 4, and $MinDist(e.loc, N.rectangle)$ is the minimum Euclidian distance between e.loc and any point on $N.rectangle$.

Definition 6. ***UB*(*e*, *N*):** *Given an event e and a node N, according to Eq. 1, the total upper bound $UB(e, N)$ is defined as follows:*

$$\begin{aligned} UB(e,N) &= \max_{\alpha \in [N.\alpha_{min}, N.\alpha_{max}]} \min(1-\alpha,\ UB_t(e,N)) + \alpha \cdot UB_s(e,N) \\ &= \max_{\alpha \in [N.\alpha_{min}, N.\alpha_{max}]} \min(\alpha \cdot (UB_s(e,N)-1)+1,\ \alpha \cdot UB_s(e,N) + UB_t(e,N)) \end{aligned} \tag{7}$$

where $\alpha \in [N.\alpha_{min}, N.\alpha_{max}]$. Let $f_1(\alpha) = \alpha \cdot (UB_s(e,N)-1)+1$ and $f_2(\alpha) = \alpha \cdot UB_s(e,N) + UB_t(e,N)$. Since $UB_s(e,N) \leq 1$ and $\alpha \geq 0$, then $f_1(\alpha)$ is a monotone decreasing function and $f_2(\alpha)$ is a monotone increasing function. Thus, we can get a more succinct formula to calculate $UB(e,N)$.

$$UB(e,N) = \begin{cases} 1 - N.\alpha_{min} + N.\alpha_{min} \cdot UB_s & N.\alpha_{min} \geq (1-UB_t) \\ (1-UB_t) \cdot UB_s + UB_t & N.\alpha_{min} < (1-UB_t) < N.\alpha_{max} \\ N.\alpha_{max} \cdot UB_s + UB_t & N.\alpha_{max} \leq (1-UB_t) \end{cases} \tag{8}$$

Theorem 1. *Given an event e and a node N whose rectangle encloses a set of subscriptions $S = \{s_i, 1 \leq i \leq n\}$, we have:*

$$\forall s \in S,\ \psi(e,s) \leq UB(e,N) \tag{9}$$

Proof. Since subscription s is enclosed in the rectangle of node N, the minimum Euclidian distance between $e.loc$ and any point on $N.rectangle$ is no larger than the Euclidian distance between $e.loc$ and $s.loc$, i.e.:

$$MinDist(e.loc, N.rectangle) \leq dist(e.loc, s.loc) \tag{10}$$

Thus, the spatial similarity between e and s is no larger than the upper bound of spatial similarity between e and node N according to the Eqs. 4 and 6, i.e.:

$$\psi_s(e,s) \leq UB_s(e,N) \tag{11}$$

Meanwhile, the set of attributes in $N.\Gamma$ is the union of all subscriptions in node N. And for each attribute appears in both event e and $N.\Gamma$, let $e.attr_i = N.\Gamma.attr_j$, the value $N.\mathcal{T}_{attr_j}(e.val_i)$ is the maximum weight of all subscriptions in node N on that attribute. Thus:

$$(1-s.\alpha) \cdot \psi_t(e,s) \leq \min(1-s.\alpha, UB_t(e,N)) \tag{12}$$

Since $s.\alpha \in [N.\alpha_{min}, N.\alpha_{max}]$, according to Eqs. 1, 7, 11 and 12, we can get:

$$\begin{aligned} \psi(e,s) &= (1-s.\alpha) \cdot \psi_t(e,s) + s.\alpha \cdot \psi_s(e,s) \\ &\leq \min(1-s.\alpha, UB_t(e,N)) + s.\alpha \cdot UB_s(e,N) \\ &\leq \max_{\alpha \in [N.\alpha_{min}, N.\alpha_{max}]} \min(1-\alpha,\ UB_t(e,N)) + \alpha \cdot UB_s(e,N) \\ &= UB(e,N) \end{aligned} \tag{13}$$

□

Algorithm 1. R^I-treeMatch(e, $tree$, k)

```
1: queue ← new PriorityQueue(); /*The higher the value of UB, the greater the
   priority*/
2: heap ← new Min-Heap(); /*Store candidate top-k subscriptions*/
3: queue.push(tree.root, 1);
4: while not queue.empty() do
5:    element ← queue.top();
6:    queue.pop();
7:    if element.Node is a leaf node then
8:       for each entry sub ∈ element.Node do
9:          if heap.size() < k then
10:            heap.insert(sub, ψ(e, sub));
11:         else if ψ(e, sub) > heap.begin().key then
12:            heap.erase(heap.begin());
13:            heap.insert(sub, ψ(e, sub))
14:      if heap.size()==k and
         (queue.empty() or heap.begin().key ≥ queue.top().key) then
15:         break;
16:   else
17:      for each entry node ∈ element.Node do
18:         queue.push(node, UB(e, node));
19: reverse all elements in heap and return;
```

5.2 Matching

Now, we discuss how to use the R^I-tree to solve SM-k problem when an event e arrives. The best-first traversal algorithm (e.g., [15]) is used to retrieve the top-k best matched subscriptions. A priority queue is used to store the nodes that have yet to be visited (i.e., the node with higher UB has a greater priority). And a global min-heap of size $O(k)$ is maintained and is used to store top-k subscriptions among all subscription objects visited.

Algorithm 1 shows the pseudocode of retrieving top-k subscriptions on the R^I-tree. The algorithm always picks the node N with the largest $UB(e, N)$ value in the priority queue. The algorithm terminates when k subscriptions have been found and the similarity of the smallest one is not smaller than the largest one in the priority queue (or the priority queue is empty).

6 Evaluation

In this section, we evaluate the R^I-tree based solution by conducting extensive experiments on a very large data set. All algorithms are implemented in C++ and compiled using g++ 4.2.1 and the experiments are run on a 2.3 GHz Intel(R) Core(TM) core i7 processor with 16 GB of *RAM*.

Table 4. Experimental Parameters

Param	Description	Value
N_e	# Events	1000
N_s	# Subscriptions	1M, **2M**, 3M, 4M, 5M
d	# Dimensions	50, 100, **400**, 600, 800
c	Dimension cardinality	10, 50,**100**,250,500
L_e	Avg. event length	6, 8, **10**, 12, 14
L_s	Avg. subscription length	2, 3, **4**, 5, 6
k	Top-k parameter	1, 3, **5**, 7, 9
f_α	Distribution of α	$\mathcal{N}(0.1, 0.05)$, $\mathcal{N}(0.5, 0.05)$, $\mathcal{N}(0.9, 0.05)$, $\boldsymbol{\mathcal{U}(0, 1)}$

6.1 Experimental Setup

We evaluate the following algorithms: (1) SCAN(a sequential scan); (2) R^t-tree (we extend R^t-tree [19] to support our model, i.e., constructing an R^t-tree, changing its *TokenSet* to the attribute set which includes attribute id, range and maximum weight and traversing the R^t-tree from the root to leaves); (3) TAS (the threshold algorithm based solution); (4) R^I-tree (the R^I-tree based solution). In the experiments, all subscriptions are first loaded into the memory and indexed by corresponding index structures, and then events are read as input continuously. We record and analyze the average time and space cost of matching top-k matched subscriptions with an event. The maximum number of children of a node in the R^I-tree and the R^t-tree is 50 in our experiments.

Since there is no suitable public real subscription and event dataset for top-k subscription matching, we use a synthetic dataset by combining non-spatial data generated by BE-Gen[5] [16] and spatial data selected from real twitter data with location information in USA [3] for both events and subscriptions. For each boolean expression generated by BE-Gen, we randomly assign a location to it from the twitter dataset.

As to the weight for each predicate of subscriptions, we adopt the weight generation technique proposed in [18] (this way is also used in [17]): (i) For each unique attribute $attr$, first compute its reciprocal of frequency, denoted by ξ_{attr}, based on the concept that popular attribute should be assigned a low weight while an infrequent attribute should be assigned a high weight. (ii) For each predicate $p_i = (attr, op, val, \omega)$, its weight is computed as follows: $p_i.\omega = \max(\xi_{p_i.attr},\ x)$, where x is randomly generated from a Gaussian distribution: $\mathcal{N}(0.8 \times \xi_{p_i.attr}, 0.05 \times \xi_{p_i.attr})$.

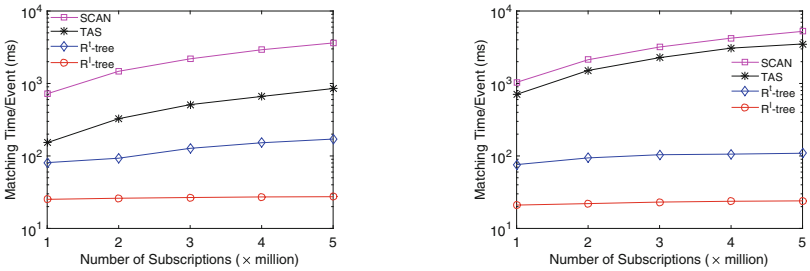
In our experiments, we evaluate the performance of those algorithms under different data distributions(Uniform and Zipf). Table 4 summarizes the main parameters used in experiments (default values are in bold).

[5] http://msrg.org/datasets/BEGen.

6.2 Experimental Results and Analyses

We conduct 6 groups of experiments, observing the effect of the number of subscriptions, the space dimensionality, the dimension cardinality, the average subscription/event length, the top-k parameter and the distribution of α on matching time. In each group of experiments, we conduct the experiments under different distributions of choosing predicates' attributes (Uniform and Zipf).

Varying the Number of Subscriptions. In the first group of experiments, we observe the effect of the number of subscriptions. Figure 5 show the performance of 4 algorithms when the number of subscriptions is varied from 1 million to 5 millions under different workload distributions. In general, with the increase of the number of subscriptions, the matching times of R^t-tree, TAS, and SCAN all increase quickly while our solution is very smooth. In these experiments, the matching time of R^I-tree on average is 4.8 and 4.3 times faster than the next best algorithm for the uniform distribution and the Zipf distribution, respectively.

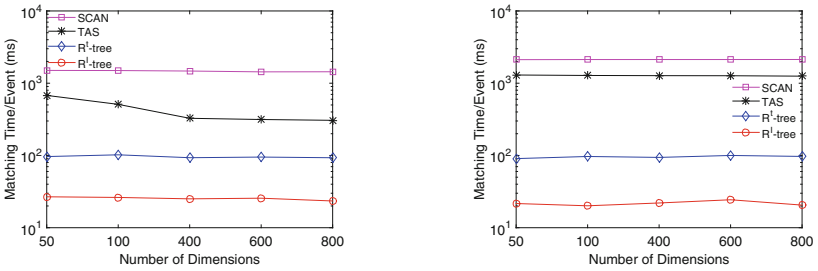


(a) Unif (b) Zipf

Fig. 5. Varying the number of subscriptions

Varying Space Dimensionality. Compared with the effect of varying the number of subscriptions, the effect of space dimensionality is slight. All algorithms with the exception of TAS are almost unchanged as the dimensionality varies, as shown in Fig. 6. In Fig. 6a, under the uniform distribution the matching time of TAS decreases as the dimensionality increases, since subscriptions tend to share less common predicates when the dimensionality is large. However, under the Zipf distribution the matching time of TAS does not increase as the dimensionality increases, it is because there are a few popular dimensions among all subscriptions, leading to a large of overlap among subscriptions. Overall, on average R^I-tree is 3.7 and 4.4 times faster than the next best algorithm for the uniform distribution and the Zipf distribution, respectively.

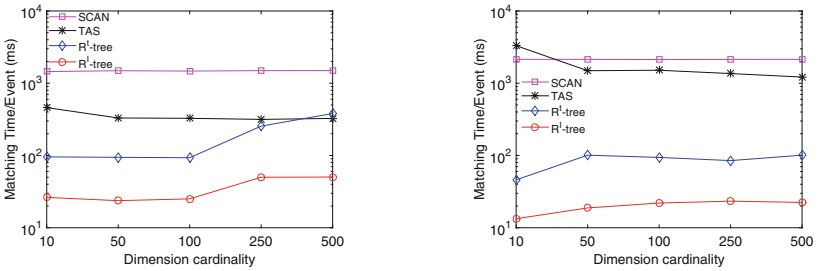
Varying the Dimension Cardinality. In this group of experiments, we observe the effect of the dimension cardinality. When the dimension cardinality increases, the matching rate between an event and all subscriptions will decrease.



(a) Unif (b) Zipf

Fig. 6. Varying space dimensionality

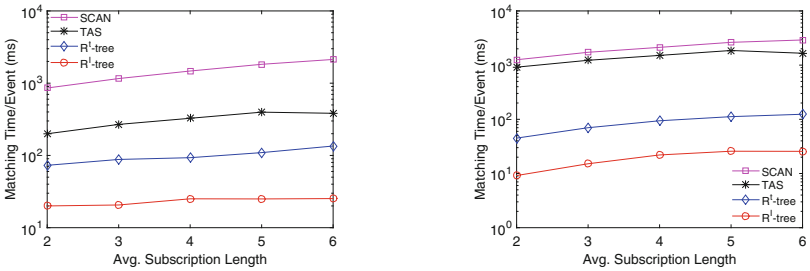
Facing such data set, the R-tree based algorithms will visit more nodes, resulting in the increasing of matching time. However, as shown in Fig. 7, on average, R^I-tree is still 4.8 and 4.2 times faster than R^t-tree for the uniform distribution and the Zipf distribution, respectively.



(a) Unif (b) Zipf

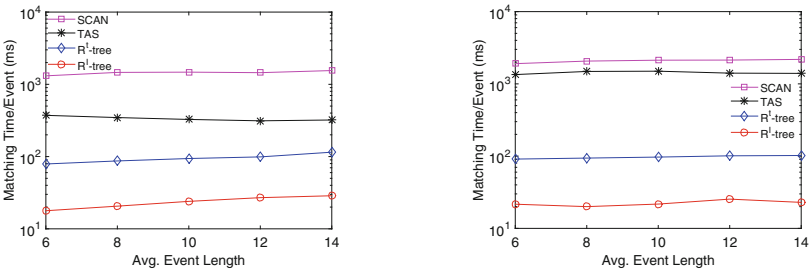
Fig. 7. Varying dimension cardinality

Varying Average Subscription/Event Length. Another key factor which can affect the algorithm performance is the average number of predicates per subscription and event. As shown in Fig. 8, all algorithms are sensitive to the average subscription length. It is because the overlap among subscriptions increases. Compared with the average subscription length, 4 algorithms are insensitive to the average event length, as shown in Fig. 9. The matching times of R^I-tree and R^t-tree both increase since the upper bound of non-spatial similarity will increase as the average event length increases. In general, R^I-tree performs best because of its filtering strategy. It is 4.3 and 4.6 times faster than the next best algorithm for the uniform distribution and the Zipf distribution, respectively, as the average subscription length increases. It is also 4 and 4.4 times faster than the next best algorithm for the uniform distribution and the Zipf distribution, respectively, as the average event length increases.



(a) Unif (b) Zipf

Fig. 8. Varying average subscription length

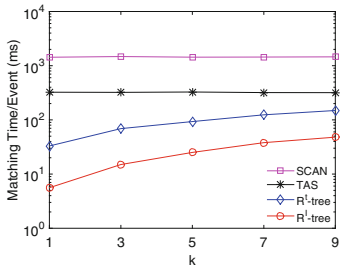


(a) Unif (b) Zipf

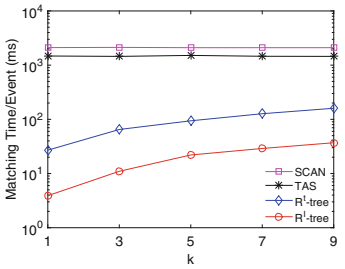
Fig. 9. Varying average event length

Varying the Value of k. Now we observe the effect of the top-k parameter, here, the value of k grows from 1 to 9. Figure 10 shows the performance of 4 algorithms when varying k under different workload distributions. In general, with the increase of k, the matching times of R^t-tree and R^I-tree increase marginally. However, R^I-tree still has the best performance. For instance, at $k = 9$, under the uniform distribution, the matching time of R^I-tree is 3, 6, and 30 times better than R^t-tree, TAS, and SCAN, respectively. Similarly, under the Zipf distribution, the speed-up ratios are 4, 39, and 57 times, respectively.

Varying the Distribution of α. Finally, we conduct a group of experiments observing the effect of the distribution of α. Figure 11 shows the performance of 4 algorithms under different distribution of α, i.e., Normal distribution :$\mathcal{N}(0.1, 0.05)$, $\mathcal{N}(0.5, 0.05)$ and $\mathcal{N}(0.9, 0.05)$; Uniform distribution $\mathcal{U}(0, 1)$. Under Normal distribution, the R^I-tree performs much better than all other algorithms and also is better than the situation R^I-tree at $\mathcal{U}(0, 1)$. The reason behind this is that the values of α for all subscriptions are close to the mean and most search paths can be filtered efficiently by the pruning strategy. For example, at $\mathcal{N}(0.5, 0.05)$, the matching time of R^I-tree is 21 and 133 times faster than the next best algorithm (TAS) for the uniform distribution and the Zipf distribution, respectively.

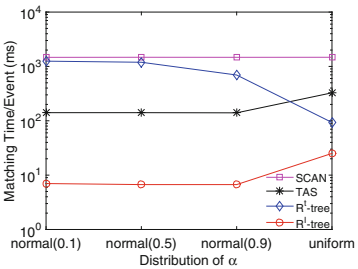


(a) Unif

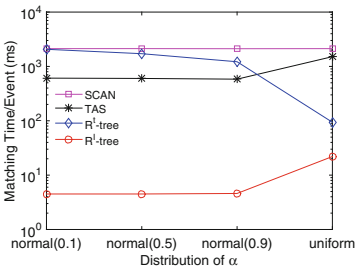


(b) Zipf

Fig. 10. Varying k



(a) Unif



(b) Zipf

Fig. 11. Varying the distribution of α

7 Conclusions

In this paper, we propose and formalize a variant of top-k subscription matching, i.e., top-k subscription matching for location-aware Pub/Sub systems which supports boolean expressions in subscriptions. We propose a novel index structure R^I-tree, which combines the R-tree and the dynamic interval-tree. In addition, we develop an efficient filtering strategy to reduce the search space. Finally, we evaluate the R^I-tree based solution by experiments on a large-scale dataset. The experimental results convincingly demonstrate the benefits of our algorithm.

Acknowledgments. Jiafeng Hu and Reynold Cheng were supported by the Research Grants Council of Hong Kong (RGC Project (HKU 711110)). Dingming Wu was supported by HKU 714712E. Beihong Jin was supported by the National Natural Science Foundation of China under Grant No. 61472408 and the Opening Foundation of Beijing Key Lab of Intelligent Telecommunications Software and Multimedia, Beijing University of Posts and Telecommunications.

References

1. Chen, L., Cong, G., Jensen, C.S., Wu, D.: Spatial keyword query processing: an experimental evaluation. In: PVLDB'2013, pp. 217–228. VLDB Endowment (2013)
2. Chen, L., Cong, G., Cao, X., Tan, K.-L.: Temporal spatial-keyword top-k publish/subscribe. In: ICDE 2015, pp. 255–266 (2015)
3. Cheng, Z., Caverlee, J., Lee, K., Sui, D.Z.: Exploring millions of footprints in location sharing services. In: ICWSM 2011, pp. 81–88 (2011)
4. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. In: VLDB 2009, vol. 2, pp. 337–348 (2009)
5. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer-Verlag, Heidelberg (2000)
6. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.-M.: The many faces of publish/subscribe. ACM Comput. Surv. **35**(2), 114–131 (2003)
7. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: PODS 2001, pp. 102–113. ACM, NY, USA (2001)
8. Fontoura, M., et al.: Efficiently evaluating complex boolean expressions. In: SIGMOD 2010, pp. 3–14. ACM, NY, USA (2010)
9. Guo, L., Zhang, D., Li, G., Tan, K.-L., Bao, Z.: Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In: SIGMOD 2015, pp. 843-857. ACM (2015)
10. Guttman. A.: R-trees: a dynamic index structure for spatial searching. In SIGMOD 1984, pp. 47–57. ACM, NY, USA (1984)
11. Hu, H., Liu, Y., Li, G., Feng, J., Tan, K.-L.: A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In: ICDE 2015, pp. 711–722 (2015)
12. Kaplan, H., Molad, E., Tarjan, R.E.: Dynamic rectangular intersection with priorities. In: STOC 2003, p. 639. ACM Press, New York, June 2003
13. Li, G., Wang, Y., Wang, T., Feng, J.: Location-aware publish/subscribe. In: KDD 2013, pp. 802–810. ACM Press, New York (2013)
14. Machanavajjhala, A., Vee, E., Garofalakis, M., Shanmugasundaram, J.: Scalable ranked publish/subscribe. In: VLDB 2008, vol. 1, issue no. 1, pp. 451–462, August 2008
15. Mehlhorn, K.: Data structures and algorithms 3: Multi-dimensional Searching and Computational Geometry. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (1984)
16. Sadoghi, M., Jacobsen, H.-A.: Be-tree: an index structure to efficiently match boolean expressions over high-dimensional discrete space. In: SIGMOD 2011, pp. 637-648. ACM (2011)
17. Sadoghi, M., Jacobsen, H.-A.: Relevance matters: Capitalizing on less (top-k matching in publish/subscribe). In: ICDE 2012, pp. 786–797 (2012)
18. Whang, S.E., Garcia-Molina, H., Brower, C. J. Shanmugasundaram, S. Vassilvitskii, E. Vee, and R. Yerneni. Indexing boolean expressions. In: VLDB 2009, vol. 2, issue no. 1, pp. 37–48 (2009)
19. Yu, M., Li, G., Wang, T., Feng, J., Gong, Z.: Efficient filtering algorithms for location-aware publish/subscribe. IEEE TKDE **27**(4), 950–963 (2015)
20. Zhang, D., Chan, C.-Y., Tan, K.-L.: An efficient publish/subscribe index for e-commerce databases. In: Proceedings VLDB Endow, vol. 7, issue no. 8, pp. 613–624 (2014)
21. Zhang, D., Tan, K.-L., Tung, A.K.H.: Scalable top-k spatial keyword search. In: EDBT 2013, pp. 359–370. ACM Press, New York, USA (2013)