An Efficient Location-Aware Top-k Subscription Matching for Publish/Subscribe with Boolean Expressions

Hanhan Jiang[1], Pengpeng Zhao[1,2(✉)], Victor S. Sheng[3], Jiajie Xu[1,2], An Liu[1,2], Jian Wu[1,2], and Zhiming Cui[1,2]

[1] School of Computer Science and Technology, Soochow University, Suzhou, China
ppzhao@suda.edu.cn
[2] Collaborative Innovation Center of Novel Software Technology and Industrialization, Suzhou 215006, China
[3] Computer Science Department, University of Central Arkansas, Conway, USA
ssheng@uca.edu

Abstract. Location-aware publish/subscribe (pub/sub) has attracted a lot of attentions with the booming of mobile Internet technologies and the rising popularity of smart-phones. Subscribers subscribe their interests with their locations as subscriptions, and publishers publish geo-information as events. Many state-of-art applications with a massive amount of geo-information, such as location-aware targeted advertising systems, face this situation. Existing related work mainly focuses on unstructured geo-textual information. However, many online-to-offline applications have enormous geo-information with different structured descriptions. To handle such structured information, a new type of location-aware pub/sub approach is needed. In this paper, we handle these subscriptions using boolean expressions. Since the number of publishers and subscribers can be enormous, it is extremely important to improve the matching effectiveness and efficiency of top-k query processing. In this paper, we develop a novel solution named RR^t-trees. RR^t-trees integrates R^t-tree and a predicate index structure together to return top-k best matched subscriptions from a great number of events. Our experimental results on synthetic and real-world datasets show that RR^t-trees achieve better performance than baseline methods.

Keywords: Location-aware pub/sub · Top-k · Boolean expressions

1 Introduction

With the rapid progress of mobile Internet technologies and the growing popularity of smart phones, a great amount of geo-information is generated at an unprecedented scale. In social network applications (e.g., Facebook or Twitter), there are a great number of users. Their personal information can be described

S.B. Navathe et al. (Eds.): DASFAA 2016, Part II, LNCS 9643, pp. 335–350, 2016.
DOI: 10.1007/978-3-319-32049-6_21

by a set of attribute-value pairs and their geo-locations revealed by GPS. In a online-to-offline system, there are millions of users browsing products on the system, such products can be described by a set of attribute-value pairs and associated with geo-locations. In this paper, we refer to such data with both attribute-value pairs and geo-location information as *geo-tagged attribute-value pair objects.*

In a location-aware publish/subscribe system, subscribers subscribe their interests and publishers publish events with geo-information. This kind of systems has many real-world applications. In a location-aware targeted advertising application system, advertisers are subscribers, who specify the properties of their interested users. For example, they can have a subscription like *(e.g.,"16⩽ age⩽*28,*hobby*∈{*Tennis, basketball*}","*51.16145,0.14123")*. The system will display corresponding advertisements on users screens. This is a well-known pushing advertising model. Users of social network systems (such as Facebook, Twitter, etc.) act as publishers. Their personal information, such as age, hobby and locations, becomes an event *(e.g.,"age=20, sex=female, hobby=tennis,school=Harvard", "51.16515,0.14123")*. Advertisements can be displayed on these users screen if there is a high relevance between an event and a subscription. This kind of information pushing model is useful for online-to-offline commerce platforms, such as Groupon[1], Sellers or service providers in Groupon system are subscribers, who may want to accurately push their advertisements to potential customers by specifying both users properties and a series of their product information *(e.g.,"hobby=smart-phone, item*∈{*Iphone6s, Iphone6*}, *price⩽499*$", "*51.25643, 0.14845")* as a subscription. Users of this system are publishers. When a user click a product link, the information of this product and the users properties can become an event*(e.g.,"hobby=smart-phone, item=Iphone6s, price=469", "51.2612, 0.12545")*. In such applications, only a few advertisements can be displayed due to the limited screen size.

Existing unstructured location-aware publish/subscribe systems [1,3,5,8,19,20] can support subscriptions with geo-textual descriptions very well. For example, users of Twitter can register their interests with a geo-textual description *(e.g.,"Cheapest iphone6s", "31.4522,51.4451")*. The system has to ensure a timely delivery of relevant geo-textual objects to the user. However, this kind of location-aware pub/sub systems cannot support *geo-tagged attribute-value pair objects*, which need a structured description to capture attributes and values. Existing structured location-aware pub/sub systems [5,13]use a boolean expression presenting a subscription. They can efficiently retrieve all matched information. Therefore, a user may be overwhelmed. To address these issues, we propose a new type of top-k subscriptions matching with boolean expressions, referred as *Location-aware Top-k Subscription Matching with Boolean Expressions*(in short TSMB-loc). The latest solution proposed for top-k subscription matching with boolean expression [7] focused on fuzzy matches. We will develop a solution ensuring strict boolean semantics of expressions.

[1] http://www.Groupon.com.

There are two challenges on developing solutions for location-aware top-k subscription matching with boolean expressions. First, how can we filter out the candidates of top-k best subscriptions from millions of subscriptions with a large amount of attributes, values and geo-locations. Second, it needs to retrieve the top-k best subscriptions over tons of candidates. Thus, an efficient and effective solution to cope with the TMSB-loc problem is necessary.

To efficiently and effectively process the TSMB-loc problem, we propose a novel R^t-tree based index structure, ranked R^t-tree (called RR^t-trees since then), by integrating the R^t-tree [3] index structure with a predicate index structure together and using a subscription partitioning scheme. When an event with location information arrives, our method can quickly retrieve its top-k best matched subscriptions. To summarize, we make the following contribution:

- We propose a new problem, location-aware top-k subscription matching with boolean expressions(TSMB-loc).
- We propose a new index structure, called RR^t-trees as the solution of TSMB-loc, which can efficiently retrieve top-k best subscriptions from millions of subscriptions.
- We conduct experiments on a synthetic dataset and a real-world dataset to evaluate the performance of our proposed RR^t-trees.

The remaining of this paper is organized as follows. In Sect. 2, we overview related work. Then, we formalize the problem in Sect. 3. In Sect. 4, we propose a baseline solution by extending a boolean expression index structure and a state-of-the-art spatial index R-tree. In Sect. 5, we propose an advanced solution, the RR^t-trees index structure. In Sect. 6 we give the similarity upper bound of RR^t-trees. In Sect. 7, we present the matching algorithm of RR^t-trees. Extensive experimental results are reported in Sect. 8, and we conclude this paper in Sect. 9.

2 Related Work

This research topic is closely related to two main research branches: structured pub/sub systems and location-aware unstructured pub/sub systems. We will briefly review these two branches as follows.

Structured Pub/Sub. There are some researches on structured pub/sub with boolean expressions [2,4,9,11,12,18,21]. Guo et al. [5] proposed a new location-aware pub/sub system named *Elaps*. *Elaps* can continuously detect moving subscriptions of users over event streams. Hu et al. [7] proposed a R^I-tree for top-k subscription matching over structured information. They are all different to our problem since *Elaps* cannot maintain a top-k best matched subscriptions and R^I-tree is a partial matching in a boolean expression. To adapt to different workloads, Sadoghi and Jacobsen [10] presented BE*-tree index structure. BE*-tree allows the values of attributes to be continuous. It combines a bi-directional tree expansion mechanism with an overlap-free splitting strategy. D et al. [21] proposed Op-index, which builds an inverted index over the pivot attributes

of subscriptions and developed a two-level partitioning scheme to handle subscriptions with high dimensional attributes. However, BE*-tree does not make the location dimension into consideration and Op-index can not return top-k subscriptions.

Location-Aware Unstructured Pub/Sub. There are many related researches on unstructured location-aware pub/sub over geo-textual data. [1,5,6,8,14–17,19,20,22–24]. To study the location-aware pub/sub problem for parameterized spatio-textual subscriptions, Hu et al. [6] presented a filter-verification framework by integrating prefix filtering and spatial pruning techniques together. To efficiently filter geo-textual data, Li et al. [8] proposed R^t-tree,which loads the selected token from subscriptions into different R-tree nodes. To support ranking semantics, Yu et al. [20] make an extension of R^t-tree. These three works [6,8,20] only focus on geo-textual information, which cannot retrieve top-k subscription matching. Note that pub/sub focusing on geo-textual cannot support geo-tagged attribute-value pair objects. We propose RR^t-trees, which has two distinguishing features. First, RR^t-trees allows user to specify their interests in form of boolean expressions. A boolean expression is much more expressive than that of a textual content. Second, RR^t-trees focus on retrieving top-k best matched subscriptions.

3 Problem Definition

In this section we formally define the problem of location-aware top-k subscription matching with boolean expressions.

Subscription: Subscribers register their interests as subscriptions. A subscription s is consisted by three elements: $s.B$, $s.loc$, α, where $s.B$ is a boolean expression to describe the interests of subscribers, $s.loc$ is the spatial location of a subscriber, and α is a parameter to balance the relative importance between spatial similarity and boolean expression similarity (we call it BE similarity since then). The boolean expression is a combination of predicates in conjunctive normal form. A predicate is a constraint specified by users to represent the relationship between an attribute and its value. A predicate contains three elements: an attribute A, an operator f_{op}, and a value v. That is, p(A,f_{op},v) denotes a predicate p. The operator can be a relational operator ($<$,$>$,$\leqslant$,$\geqslant$,$=$,$\neq$), or a set ($\in$,$\notin$). Each predicate has a weight ω_s, where $\sum_{i=1}^{n} \omega_{si} = 1$. Thus, the subscription can be modeled as follows:

$$s : \{[(p_1, \omega_{s1}) \wedge (p_2, \omega_{s2}) \wedge (p_i, \omega_{si}) \wedge ... \wedge (p_n, \omega_{sn})],\ \text{loc}, \alpha\}$$

Event: An event e contains a collection of attribute-value pairs denoted as $e.V$ and a geo-position denoted as $e.loc$. The attribute-value pairs $e.V$ are represented in the form of conjunction of predicates with equality operator. That is, $\nu(A, v)$ denotes an attribute-value pair ν. Each attribute-value pair has a weight ω_e, where $\sum_{i=1}^{n} \omega_{ei} = 1$. Thus, an event can be denoted as follows:

$$e : \{[(\nu_1, \omega_{e1}) \wedge (\nu_2, \omega_{e2}) \wedge (\nu_3, \omega_{e3}) \wedge ... \wedge (\nu_n, \omega_{en})], \text{loc}\}$$

The weight ω_s is given by subscribers and is used to represent users' preference among predicates in a subscription. The weight ω_e signifies the relevance between value-pair and its predicate. It is generated according to the appearing frequency of the attribute-value pair in the whole dataset.

Definition 1. *(Predicate Match) Given an attribute-value pair ν, for a predicate p appears in a subscription, we said that there is a predicate match if $p.A = \nu.A$ and $p_i(\nu_i.v)$=true.*

Definition 2. *(Boolean Expression Match) A boolean expression $s.B$ is said to match a collection of attribute-value pairs $e.V$ if each of the predicates in $s.B$ has a match in $e.V$.*

Definition 3. *(Similarity Function ϕ) Given an event e=e:{[(ν_1, ω_{e1})∧ (ν_2, ω_{e2})∧(ν_3, ω_{e3})∧...∧(ν_n, ω_{en})], loc} and a subscription s=s: {[(p_1, ω_{s1})∧ (p_2, ω_{s2})∧ (p_i,ω_{si})∧...∧ (p_n,ω_{sn})], loc, α}, we define the similarity function $\phi(e, s)$ as follows:*

$$\phi(e, s) = s.\alpha \cdot \varphi_{BE}(e, s) + (1 - s.\alpha) \cdot \varphi_s(e, s) \tag{1}$$

where φ_{BE} is a BE similarity function and the φ_S is a spatial similarity function.

$$\varphi_{BE}(e, s) = \sum_{p_j \in s, \nu_i \in e, p_j.A = \nu_i.A, p_i(\nu_i.v) = true}^{n=s.\varsigma} \omega_{sj} \cdot \omega_{ei} \tag{2}$$

where $s.\varsigma$ is the size of subscription s(number of predicates in a subscription).

The spatial similarity is given by:

$$\varphi_s(e, s) = 1 - \frac{distance(e.loc, s.loc)}{MaxDistance} \tag{3}$$

where distance(e.loc,s.loc) is the Euclidian distance between s and e, and the MaxDistance is the maximum distance among subscriptions.

As shown in Fig. 1 for the event e={A=3(0.1)∧ B=3(0.5)∧ C=4(0.2)∧ F=2(0.2),e.loc}, the boolean expression subscription S_1 matches the attribute-value pairs of e according to Definition 2. However, the subscription boolean expression S_4 does not match attribute-value pairs of e as there is no attribute-value pair in e that matches the predicate G⩾4. Based on Definition 3, the spatial similarity φ_S(e,s_1) is 0.35, and the BE similarity φ_{BE}(e,s_1) is 0.25. Therefore, the balanced similarity ϕ(e,s_1) is 0.30. Similarly, since the spatial similarity φ_S(e,s_9) is 0.15, and the BE similarity is φ_{BE}(e,s_9) is 0.18, ϕ(e,s_9) is 0.18. Thus, if we want to retrieve top-1 best subscription for event e, the answer is S_1.

Location-Aware Top-k Subscription Matching: Given a set of subscriptions S, TSMB-loc aims to finds the top-k most relevant strict matched subscriptions $S^k \in S$. For any subscription s$\in S^k$ and $s^* \in$(S-S^k), ϕ(e,s^*)<ϕ(e,s).

S_1	B=3(0.3) ∧ A ∈ (3,2,5)(0.4) ∧ C ⩾ 2(0.3)	0.5
S_2	D ⩽ 2(0.6) ∧ C=7(0.4)	0.6
S_3	E=2(0.3) ∧ A=4 (0.6) ∧ B ⩽ 2(0.1)	0.8
S_4	A=3(0.3) ∧ F ∈ (5,2,1) (0.4) ∧ G ⩾ 4(0.3)	0.3
S_5	F ⩽ 5(0.5) ∧ D=2(0.2) ∧ F=5(0.1) ∧ G=4(0.2)	0.5
S_6	E=3 (0.3) ∧ A=5(0.1) ∧ C ⩾ 2(0.6)	0.7
S_7	E ⩾ 3 (0.5) ∧ B ∈ (3,2,5)(0.3) ∧ C=4(0.2)	0.5
S_8	A ⩽ 12(0.4) ∧ B=3(0.2) ∧ C ⩾ 2(0.3) ∧ E ⩾ 3(0.1)	0.4
S_9	A ⩽ 3(0.5) ∧ B ⩽ 3 (0.1) ∧ C=5(0.1) ∧ F ⩽ 2(0.3)	0.2
S_{10}	E=3(0.7) ∧ B ∈ (3,2,5)(0.2) ∧ C=4(0.1)	0.9
e	A=3 (0.1) ∧ B=3(0.5) ∧ C=4 (0.2) ∧ F=2(0.2)	

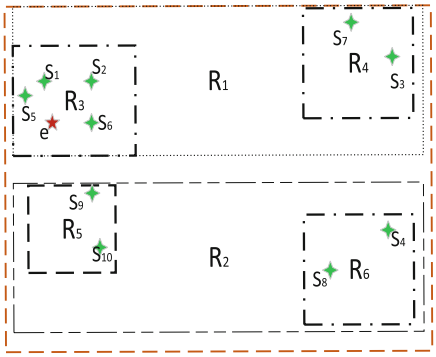


Fig. 1. An example of subscriptions and events

4 A Baseline Solution

In this section, we extend two state-of-art index structures (Op-index and R-tree) to cope with the TMSB-loc problem. These extensions will be used as baseline solution to evaluate our advanced solutions proposed in Sect. 5.

Op-index is a well-known pub/sub index structure for boolean expressions, which builds an inverted index structure over a pivot attribute[2] and designs a two level partitioning scheme to handle the pub/sub problem with high-dimensions attribute. Based on op-index and a well-known spatial information index structure R-tree, we can integrate op-index with R-tree together (called OPR-tree in short) to cope with the TSMB-loc problem. We first build an R-tree for all the locations of subscriptions. When the subscriptions fall into leaf nodes of the R-tree, we organize subscriptions inside each leaf node using the Op-index structure. The construction and the query processing of OPR-tree will be explained in details.

OPR-Tree Construction: For each subscription s, we retrieve its leaf node n in the R-tree using its spatial point information $s.loc$ and then select its pivot attribute δ_A. We partition subscriptions inside each leaf node into groups according to their pivot attributes. We can present a list of subscriptions with the same pivot attribute δ_A as a list denoted as L(n,δ_A). Each list (e.g., L(n,δ_A)) can be further partitioned based on the operators ($<$,$>$,$\leqslant$,$\geqslant$,$=$,$\neq$) of predicates. The predicates with the same operator are organized into a sub-list. A sub-list of L(n,δ_A) can be presented as L(n,δ_A,op), where *op* is a specific operator. For each group of predicates L(n,δ_A,op), we use a signature segment to map the predicates by a hash function. We compute the hash value of each predicate using a hash function h(p.A) to select a bit from the signature segment and set this bit to 1. Besides, there is a collection of counter arrays, corresponding with each subscription list L(n,δ_A,op). The value of the counter array is initialed as the

[2] The attribute with the least appearing frequency in the whole dataset becomes the pivot attribute.

size of the subscriptions. For each predicate in L(n,δ_A,op), there is a pointer to point to its corresponding counter array value.

OPR-Tree Query Processing: For an event e, we search the R-tree using the spatial point information $e.loc$ to find a corresponding leaf node n. Then we extract each candidate pivot attribute ν_i.A ($\nu_i \in$ e.V) from the set of distinct attribute-value pairs $e.V$. If $\nu_i.A$ is indeed a pivot attribute, we extract each attribute-value pair ν_i to search the predicate lists L(n,δ_A,op) in L(n,δ_A). For each attribute-value pair ν_i, we first calculate the hash value of a candidate attribute, i.e., h($\nu_i.A$). If the corresponding bit of the signature segment is 1, we search the corresponding predicate list L(n,δ_A,op). Then, the BE similarity φ_{BE}(e,s) is calculated if $p_j(\nu_i.\text{v=true})$, where $p_j \in$ L(n,δ_A,op). If the corresponding value of the counter array goes to 0, we calculate the spatial similarity φ_S(e,s). The final balanced similarity ϕ(e,s) is calculated and added into the temporary result set as a candidate top-k result. The upper bound of a given leaf node n to a given event e is a balanced similarity between the minimum Euclidian distance from e to n and the maximum weight of attribute-value in e.

OPR-tree partitions the subscriptions by the region of leaf nodes and organizes these subscriptions using Op-index. However, the region of a leaf node may be very small, which results in a poor pruning ability in high spatial dimensions. Therefore, OPR-tree is not very efficient. In order to avoid this problem, we proposed RR^t-trees method in the next section.

5 RR^t-trees Solution

In this section, we present a framework that integrates R^t-tree and a predicates index structure into a new index, named RR^t-tree. Based on RR^t-tree, we develops a partitioning scheme to organize subscriptions into disjointed RR^t-trees. Finally, we introduce the upper bound of efficiently and effectively filtering out top-k best subscriptions over this framework.

5.1 RR^t-tree Index Structure

As we discussed in Sect. 4, OPR-tree is not very efficient to meet TSM-loc since the poor pruning ability in spatial dimension.

R^t-tree [3] is an unstructured location-aware pub/sub index structure, which integrates so-called high-quality representative tokens selected from subscriptions into the nodes of R-tree. Based on R^t-tree, we propose a method, ranked R^t-tree (RR^t-tree) to cope with the TSMB-loc problem. The basic idea of RR^t-tree is to convert the tokens of R^t-tree into predicates of a subscription, which will be loaded into the ancestor nodes of a leaf node in which a subscription locates. Then, predicates in each node are indexed using a predicate index structure.

***RR^t*-Tree Construction:** We build an R-tree for all the locations of subscriptions. For a given subscription s, we first extract the distinct predicate p_i, containing its weight, where $p_i \in s$. Then, we load p_i into different nodes at different levels, which is determined by the spatial location $s.loc$ of the subscription s.

Considering a built R-tree, its height is H, the size of a given subscription s is s.ς. If s.ς>H, we directly insert the last $s.\varsigma$ - H+1 predicates. If $s.\varsigma<H$, only the ancestors in the first $s.\varsigma$ level contain the corresponding predicates of s. Letting $s.p_i$ denote the i-th predicate in s, then $s.p_i$ is loaded in an ancestor node at i-th level. For each node n on i-th level, there is a set of predicates denoted as P. We build inverted lists over the attributes of the predicates in P to organize predicates with the same attribute. To track the number of matched predicates of a subscription during an event query processing, we assign a hash map M for each subscription and initial each hash value $M[s]$ to be 0. When a predicate p_i matches an attribute-value pair, we increase its corresponding hash value by 1. Based on the number of matched predicates $M[s]$, we can efficiently filter the subscription. To explain this, we have the following lemmas.

Lemma 1. Consider an event e and a node n at the i-th level. If M[s]<i, s cannot be a candidate top-k result.

Proof. *As s appears in the i-th level, it contains at least i predicates. For each node on path from the root to node n, $s.B$ must have a predicate which cannot match all attribute-value pairs in event e. According Definition 2, s cannot be the candidate top-k result of e.*

Lemma 2. Consider an event e and a node n at the i-th level, if M[s]=s.ς, s must be a candidate top-k result

Proof. *If M[s]=s.ς, all the predicates of $s.B$ are matched by e. According to Definition 2, s must be a candidate top-k subscription of e.*

Lemma 3. Consider an event e and a node n at the i-th level, if M[s]=i<s.ς, and n is a leaf node, s cannot be a candidate top-k result of e.

Proof. *Since n is a leaf node, we have M[s]=i=H<s.ς. That is, the rest $s.\varsigma$- H+1 predicates are loaded in the node n, which cannot match all attribute-value pairs in event e. According to Definition 2, s cannot be the candidate top-k result of e.*

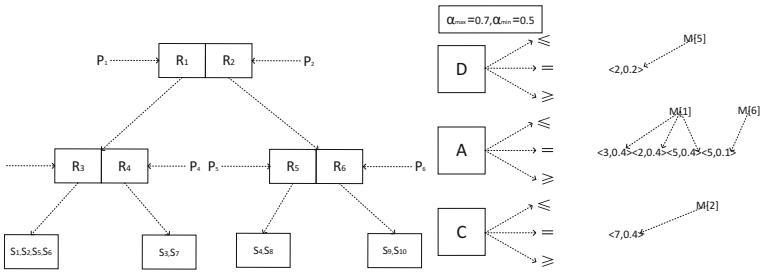
Lemma 4. Consider an event e and a node n at the i-th level. For any $p_i.A \in p_i \in P \in n$, if $p_i.A$ does not appear in e, we can directly pruning the node n.

Proof. *If $p_i.A$ does not appear in e, according to Definition 1, no predicate in node n matches any attribute-value pair. According to Definition 2, subscriptions on node n cannot be candidate top-k results of e.*

Predicate Index Structure: In each node on RR^t-tree, there are a set of predicates P, a weight of each predicate, the maximum alpha value α_{max} and minimum alpha value α_{min}. To efficiently retrieve matched predicates in P, we design an index structure for P. We index the predicates of P in two partitioning steps. In the first step, predicates are partitioned into disjointed predicate lists based on attributes as follows:

$$P = L(A_1) \cup L(A_2) \cup L(A_i) \cup ... \cup L(A_n) \tag{4}$$

For each predicate in list $L(A_i)$, there is a pointer to point to the number of matched predicates M[s] of its corresponding subscription. In second step, the



(a) RR^t-tree Index Structure (b) The predicates index structure of P_3

Fig. 2. An RR^t-tree index for the subscriptions shown in Fig. 1

predicates list $L(A_i)$ is further partitioned by their operators (we only use standard operators, such as $\leq$,$\geq$,$=$) into its corresponding value list $L(A_i, op)$ as follows:

$$L(A_i) = L(A_i, \leq) \cup L(A_i, =) \cup L(A_i, \geq) \cup ... \cup L(A_n, op) \tag{5}$$

Figure 2(a) shows the RR^t-tree index structure for the subscriptions shown in Fig. 1. Figure 2(b) shows the predicate index structure for P_3.

5.2 RR^t-trees Index Structure

Since the number of subscriptions can be very large, it is necessary to improve the efficiency of RR^t-tree. To address this problem, we partition subscriptions according to their pivot attributes (discussed in Sect. 4) into N[3] subscription lists and organize the subscription lists using disjointed RR^t-trees. We simply name this method RR^t-trees since then. Given a set of subscriptions S, we partition these subscriptions according to their pivot attributes δ_A and organize them using RR^t-trees as follows

$$S = RR^t - trees = RR^t - tree(\delta_{A1}) \cup RR^t - tree(\delta_{A2}) \cup ... \cup RR^t - tree(\delta_{An}) \tag{6}$$

From Definitions 1 and 3, we can conclude that if an event e matches a subscription s, then all the attributes in s must appear in e. Obviously, if there is an attribute in s but not in e, e wont be matched by s. Thus, given an event e, we only consider the subscriptions whose pivot attributes appear in e. Attributes with a low frequency in a whole dataset has low probabilities to appear in subscriptions. Thus, we choose the lowest frequency attribute in a subscription as the pivot attribute.

The index structure of RR-trees for the subscriptions shown in Fig. 1 is shown in Fig. 3. According to the rule of selecting pivot attributes mentioned above, A, D, E and G are selected as a pivot attribute respectively. Given an event e in Fig. 1, subscriptions in both $L(E)$and $L(G)$ don't match e definitely.

[3] The number of distinct attributes in a whole dataset.

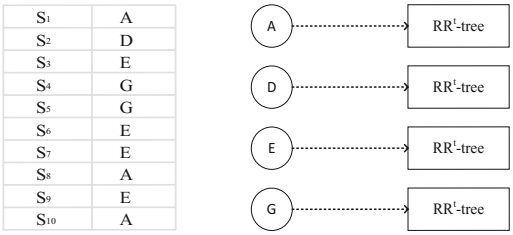


Fig. 3. The index structure of RR^t-trees

6 Similarity Upper Bound of RR^t-tree and RR^t-trees Solution

After having described the RR^t-tree and RR^t-trees index structure, it is the time to introduce the upper bound of similarity.

Definition 4. *($UB_{BE}(e,n)$) For a given event e and a node n in RR^t-tree, the upper bound of the BE similarity $UB_{BE}(e,n)$ is defined as follows. The BE similarity bound of a given event e to a node n is:*

$$UB_{BE}(e,n) = Max\left\{s \in n.parent\left[\sum_{1}^{i-1}\omega_{si}\cdot\omega_{ej} + \omega^*_{emax}\cdot\left(1-\sum_{1}^{i-1}\omega_{si}\right)\right]\right\} \quad (7)$$

where $\sum_1^{i-1}\omega_{si}\cdot\omega_{ej}$ is the total score of matched predicates of s appearing in level 1 to level i-1 where $i>1$, and ω^*_{emax} is the maximum weight of unmatched attribute-value pairs in e for subscription s. And $1-\sum_1^{i-1}\omega_{si}$ is the remaining total weights of unmatched predicates in subscription s.

Definition 5. *($UB_S(e,n)$) For a given event e and a node n, the upper bound of the spatial similarity $UB_S(e,n)$ is defined as follows:*

$$UB_S(e,n) = (1 - \frac{MinDistane(e.loc, n.MBR)}{MaxDistance}) \quad (8)$$

where MaxDistance is the maximum distance between subscriptions, n.MBR it the minimum bounding rectangle of node n and $MinDistance(e.loc, n.MBR)$ is the minimum Euclidian distance between $e.loc$ and any point on $n.MBR$.

Definition 6. *(UB(e,n)) According to Eqs. 7 and 8, for a given event e and a node n, the total upper bound UB(e,n) is defined as follows:*

$$UB(e,n) = Max\left\{\forall\alpha \in (\alpha_{min},\alpha_{max})\, min\left[1-\alpha, UB_{BE}(e,n)\right] + \alpha\cdot UB_S(e,n)\right\} \quad (9)$$

where α_{min}, α_{max} are the minimum and maximum α of subscriptions in node n.

According to Definition 6, we have the following lemma:

Lemma 5. Given an event e and a node n whose MBR encloses a set of subscriptions S^n. For any subscription s where $s \in S^n$, there is:

$$\phi(e, s) < UB\,(e, n) \tag{10}$$

We omit the proof due to space constraints.

7 Matching Algorithm

How can we use RR^t-tree to retrieve top-k best matched subscriptions with boolean expressions? We show the processing of retrieving top-k best matched subscriptions in Algorithm 1. We use a bound-queue to store the nodes that have not been visited in the algorithm. The nodes in the bound-queue is ordered by their $UB(e, n)$ in a descending order, which is calculated according to Eq. 9 from its parent node. For the root node, the bound is 1. Given an event e, we traverse all the RR^t-tree(ν_i.A) in RR^t-trees from the root node, where $\nu_i \in e$. The algorithm will return candidate subscriptions for the top-k best matched subscriptions. It will stop under two situations as follows.

- When k subscriptions are found and its minimum similarity is larger than the maximum $UB(e, n)$ in the bound-queue.
- When the bound-queue is empty.

8 Experiments

In this section, we will evaluate three indexes (OPR-tree, RR^t-tree, RR^t-trees) on synthetic and real-world datasets. All the methods are implemented in java (JDK7) and experiments are running on a machine with 3.2 GHz Intel(R) (TM) Core i5-3470 CPU and 16 GB of RAM.

8.1 Experimental Setup

Two datasets are used in our experiments, one synthetic dataset and one real-world dataset (eBay dataset), shown in Table 1. To generate the synthetic dataset, we implement a data generator, which can generate attributes, operators and values. For the set operator $\in, \notin$, we rewritten them into standard operator $=$, $\geq$,$\leq$. For the weights of attribute-value pairs in an event are generated base on this equation $\omega_{ei} = \frac{\nu.f}{\sum_{i=0}^{n} \nu_i.f}$, where $\nu_i.f$ is the frequency of an attribute-value pair in the whole dataset. And n is the number of attribute-value pairs in an event. We totally generate 5M synthetic subscriptions corresponding with 10k events for event matching tests in the synthetic dataset. For the real-world dataset (eBay), we generate 10M subscriptions and 10k events based on 10k product messages, and we extract the spatial information from Twitter to generate final subscriptions and events.

Algorithm 1. Matching(e,k)

```
1  Input An event e and the value k
2  Output Top-k best matched subscriptions S
3  Initialize :a bound-queue,a hash map M, a candidate top-k subscription list R
   and a temporary similarity storage of subscriptions Temp
4   Extract each attribute-value pair ν_j from e
5  for each RR^t-tree(δ_Ai) do
6      if (ν_j.A == δ_Ai) then
7          Search RR^t-tree(δ_Ai);
8          for each node n in RR^t-tree(δ_Ai) do
9              if (n is a root node) then
10                 bound-queue.Push(n, 1);
11                 Visit bound-queue.Pop();
12                 for each predicate p_i(ν_j.v)==true do
13                     Temp[s] =Temp[s] + ω_si · ω_ej;
14                     if (++M[s] == s.ς) then
15                         φ(e, s) = (1 − s.α) · Temp[s] + s.α · φ_S(e, s);
16                         if (R.size < k) then
17                             R.add(s);
18                         else
19                             if φ(e, s) > R.min then
20                                 R.add(s);
21                                 if (++M[s] < L) then
22                                     /*L is the i-th level*/
23                                     Temp.remove(s);
24         if (n is visited and node n is not a leaf node) then
25             bound-queue.Push(n.child,UB(e,n.child));
26         if (n is a leaf node) then
27             Visit bound-queue.Pop();
28     if (bound-queue.Empty or (R.min> bound-queue.Pop().UB and
       R.size=k)) then
29         Return R;
```

8.2 Experimental Results

In this section, we will evaluate three indexes on synthetic datasets and real-world datasets. For synthetic datasets we evaluate the performance of three indexes from different perspectives, a varied number of subscriptions and distinct attributes, varied average size of subscriptions,varied average size of events, varied parameter k and α.

Matching Time under Various k: The value of k is an important parameter for the TSMB-loc problem. Figure 4 shows the performance of the three methods

Table 1. Parameters and settings

Parameters	Synthetic dataset	eBay
Number of subscription	1M, 2M, 3M, 4M, 5M	2M, 4M, 6M, 8M, 10M
Average subscription size	4∼20	2∼10
Average event size	5 25	8
Max value of α	0.2, 0.4, 0.6, 0.8, 1.0	0.2, 0.4, 0.6, 0.8, 1.0
Number of distinct attribute	5k, 10k, 15k, 20k, 25k	10k
Top-k parameter	1, 5, 10, 20	10, 20

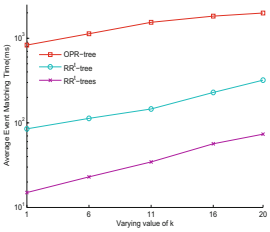


Fig. 4. Varying value of k on ebay dataset

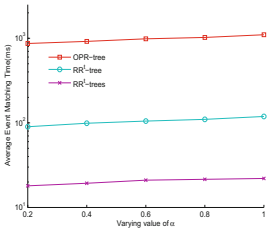


Fig. 5. Varying value of α on eBay dataset

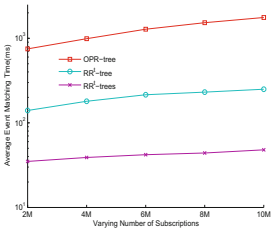


Fig. 6. Varying number of subscriptions on eBay dataset

with the increment of k on ebay dataset. We can clearly see that the average matching time of all the three methods increases with the increment of k. However, RR^t-trees achieves the best performance because of its powerful pruning ability. It is nearly 3 times faster than the next best method RR^t-tree. OPR-tree performs the worst, using the largest average matching time.

Matching Time under Various α: We also conduct several experiments to investigate the impact of varying the maximum balance value α. Our experimental results are shown in Fig. 5. Figure 5 shows that the average matching time of all the three methods grows slowly as the value of α increases. RR^t-trees still achieve the best performance over the real-world dataset.

Matching Time on Varying Number of Subscription: From Fig. 6, we can see that all the three methods are sensitive to the number of subscriptions. And RR^t-trees achieves the lowest event matching time, followed by RR^t-tree. OPR-tree performs the worst, having the highest matching time. RR^t-trees is 3.5 times faster than RR^t-tree. This is because RR^t-trees uses pivot attributes to partition subscriptions. This causes that indexing the subscriptions using RR^t-trees much more efficient than that using a single RR^t-tree. Furthermore, the upper bounds over a small number of subscriptions are easier to calculate than those over a larger scale of subscriptions, which causes the matching time of RR^t-trees grows much more smoothly than that of the other two methods.

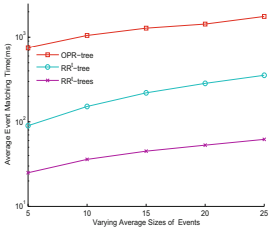


Fig. 7. Average sizes of events on synthetic dataset

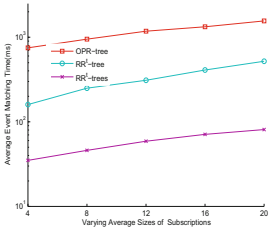


Fig. 8. Average sizes of subscriptions on ebay dataset

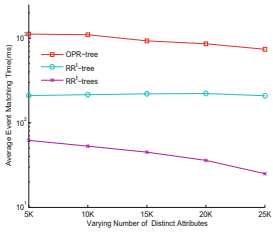


Fig. 9. Varying number of distinct attributes on synthetic dataset

Matching Time under Different Sizes of Events: Again, since the size of each event in the real-world dataset eBay is fixed, we couldnt conduct these experiments on the dataset eBay. We conduct experiments over the synthetic dataset. The experimental results are shown in Fig. 7. We can see that the average matching time of the three methods increases with the increment of the size of events. This is because the number of candidate subscriptions increases when the size of events increases. RR^t-trees scales much better than the other two methods (RR^t-tree and OPR-tree). It is about 3 times faster than the next best method RR^t-tree.

Matching Time under Different Sizes of Subscriptions: The size of subscriptions can affect the matching performance. The average event matching time on different sizes of subscriptions of the three methods on ebay dataset is reported in Fig. 8. From this figure, we can see that all the three methods are sensitive to the size of subscriptions. But, both RR^t-trees and RR^t-tree scale better than OPR-trees. It is because than both in RR^t-trees and RR^t-tree, subscriptions are pruned by each predicate on the nodes of R-tree, as we described in Sect. 5. RR^t-trees scales better than RR^t-tree because of its pruning ability of pivot attributes. However, OPR-tree only prunes subscriptions by the spatial bounds.

Matching Time under Different Numbers of Distinct Attributes: Since the number of distinct attributes is fixed in the real-world dataset eBay, we conduct experiments with various numbers of distinct attributes on our synthetic dataset. As Fig. 9 shows, when the number of attributes increases, the average matching time of both RR^t-trees and OPR-tree decreases. However, RR^t-tree performs oppositely. With the increment of number of attributes, its average matching time increases a little. $RR^t - trees$ deceases gradually, because it generates many more narrowed partitions when the number of distinct attributes increases during partitioning subscriptions according to pivot attributes.

9 Conclusion

In this paper, we tackled the problem of location-aware top-k subscription matching, which is significant for location-aware publish/subscribe systems with a

stream of geo-tagged attribute-value pair objects. Facing the challenge of efficiently and effectively delivering events to the top-k subscribers, we proposed a novel index structure called RR^t-trees, which integrates R^t-tree and a predicate index structure. In addition, we developed an efficient filtering strategy to reduce the searching space. Extensive experiments conducted in both synthetic and real-world datasets demonstrate the effectiveness and efficiency of our algorithms.

Acknowledgment. This work was partially supported by Chinese NSFC project (61472263, 61402312, 61402311), and the US National Science Foundation (IIS-1115417).

References

1. Chen, L., Cong, G., Cao, X., Tan, K.L.: Temporal spatial-keyword top-k publish/subscribe. In: 2015 IEEE 31st International Conference on Data Engineering (ICDE), pp. 255–266 (2015)
2. Cugola, G., Margara, A.: High-performance location-aware publish-subscribe on GPUs. In: Narasimhan, P., Triantafillou, P. (eds.) Middleware 2012. LNCS, vol. 7662, pp. 312–331. Springer, Heidelberg (2012)
3. Eugster, G.: Location-based publish/subscribe. In: 2013 IEEE 12th International Symposium on Network Computing and Applications, pp. 279–282 (2005)
4. Fontoura, M., Sadanandan, S., Shanmugasundaram, J., Vassilvitski, S., Vee, E., Venkatesan, S., Zien, J.: Efficiently evaluating complex Boolean expressions. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 3–14. ACM (2010)
5. 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: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 843–857. ACM (2015)
6. Hu, H., Liu, Y., Li, G., Feng, J., Tan, K.L.: A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. ICDE **2015**, 711–722 (2015)
7. Hu, J., Cheng, R., Wu, D., Jin, B.: Efficient top-k subscription matching for location-aware publish/subscribe. In: Claramunt, C., Schneider, M., Wong, R.C.-W., Xiong, L., Loh, W.-K., Shahabi, C., Li, K.-J. (eds.) SSTD 2015. LNCS, vol. 9239, pp. 333–351. Springer, Heidelberg (2015)
8. Li, G., Wang, Y., Wang, T., Feng, J.: Location-aware publish/subscribe. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 802–810. ACM (2013)
9. Machanavajjhala, A., Vee, E., Garofalakis, M., Shanmugasundaram, J.: Scalable ranked publish/subscribe. Proc. VLDB Endow. **1**(1), 451–462 (2008)
10. Sadoghi, M., Jacobsen, H-.A.: Relevance matters: Capitalizing on less (top-k matching in publish/subscribe). In: 2012 IEEE 28th International Conference on Data Engineering, pp. 786–797 (2012)
11. Sadoghi, M., Burcea, I., H.a.J: Gpx-matcher: A generic Boolean predicate-based xpath expression matcher. In: EDBT 2011, pp. 45–56 (2011)

12. Sadoghi, M., Jacobsen, H.-A.: Be-tree: an index structure to efficiently match Boolean expressions over high-dimensional discrete space. In: ACM Conference on Management of Data, pp. 637–648 (2011)
13. Sadoghi, M., Jacobsen, H.A.: Location-based matching in publish/subscribe revisited. In: Proceedings of the Posters and Demo Track, p. 9. ACM (2012)
14. Shang, S., Deng, K., Xie, K.: Best point detour query in road networks. pp. 71–80. In: ACM (2010)
15. Shang, S., Ding, R., Yuan, B., Xie, K., Zheng, K., Kalnis, P.: User oriented trajectory search for trip recommendation. In: 15th International Conference on Extending Database Technology, EDBT 2012, pp. 156–167 (2012)
16. Shang, S., Ding, R., Zheng, K., Jensen, C.S., Kalnis, P., Zhou, X.: Personalized trajectory matching in spatial networks. VLDB J. **23**(3), 449–468 (2014)
17. Shang, S., Yuan, B., Deng, K., Xie, K., Zheng, K., Zhou, X.: PNN query processing on compressed trajectories. Geoinformatica **16**(3), 467–496 (2012)
18. Whang, S.E., Garcia-Molina, H., Brower, C., Shanmugasundaram, J., Vassilvitskii, S., Vee, E., Yerneni, R.: Indexing Boolean expressions. Proc. VLDB Endow. **2**(1), 37–48 (2009)
19. Xiang Wang, Y.Z., Xuemin Line, W.W.: Ap-tree: Efficiently support continuous spatial-keyword queries over stream. In: 2015 IEEE 31st International Conference on Data Engineering (ICDE), pp. 1107–1118 (2015)
20. Yu, M., Li, G., Wang, T., Feng, J., Gong, Z.: Efficient filtering algorithms for location-aware publish/subscribe. IEEE Trans. Knowl. Data Eng. **27**(4), 950–963 (2015)
21. Zhang, D., Chan, C.Y., Tan, K.L.: An efficient publish/subscribe index for e-commerce databases. Proc. VLDB Endow. **7**(8), 613–624 (2014)
22. Zheng, B., Yuan, N.J., Zheng, K., Xie, X., Sadiq, S., Zhou, X.: Approximate keyword search in semantic trajectory database. In: 2015 IEEE 31st International Conference on Data Engineering (ICDE), pp. 975–986. IEEE (2015)
23. Zheng, K., Huang, Z., Zhou, X., et al.: Discovering the most influential sites over uncertain data: a rank based approach. IEEE Trans. Knowl. Data Eng. **99**, 1 (2011)
24. Zheng, K., Zhou, X., Fung, P.C., Xie, K.: Spatial query processing for fuzzy objects. VLDB J. **21**, 729–751 (2012)