Efficient Filtering Algorithms for Location-Aware Publish/Subscribe

Minghe Yu, Guoliang Li, *Member, IEEE*, Ting Wang, Jianhua Feng, and Zhiguo Gong

Abstract—Location-based services have been widely adopted in many systems. Existing works employ a pull model or user-initiated model, where a user issues a query to a server which replies with location-aware answers. To provide users with instant replies, a push model or server-initiated model is becoming an inevitable computing model in the next-generation location-based services. In the push model, subscribers register spatio-textual subscriptions to capture their interests, and publishers post spatio-textual messages. This calls for a high-performance location-aware publish/subscribe system to deliver publishers' messages to relevant subscribers. In this paper, we address the research challenges that arise in designing a location-aware publish/subscribe system. We propose an R-tree based index by integrating textual descriptions into R-tree nodes. We devise efficient filtering algorithms and effective pruning techniques to achieve high performance. Our method can support both conjunctive queries and ranking queries. We discuss how to support dynamic updates efficiently. Experimental results show our method achieves high performance which can filter 500 messages in a second for 10 million subscriptions on a commodity computer

1 Introduction

Location-based services (LBS) have attracted significant attention from both industrial and academic communities. Many LBS services such as Foursquare (foursquare.com) and Google Maps (maps.google.com) have been widely accepted because they can provide users with location-aware experiences. Existing LBS systems employ a pull model or user-initiated model [8], [16], where a user issues a query to a server which responds with location-aware answers. For example, if a mobile user wants to find seafood restaurants nearby, she issues a query "`seafood restaurant`" to an LBS system, which returns answers based on user's location and keywords.

To provide users with instant replies, a push model or server-initiated model is becoming an inevitable computing model in next-generation location-based services. In the push model, subscribers register spatio-textual subscriptions to capture their interests, and publishers post spatio-textual messages. This calls for a high-performance location-aware publish/subscribe system to deliver messages to relevant subscribers. This computing model brings new user experiences to mobile users, and can help users retrieve information without explicitly issuing a query.

There are many real-world applications using location-aware publish/subscribe services. The first one is Groupon. Groupon customers register their interests with locations and keywords (e.g., "`iphone4s`" at NewYork). For each Groupon message (e.g., "`iphone4s AT&T package`" at `Manhattan`), the system provider sends the message to the customers who may be potentially interested in the message by evaluating the spatial proximity and textual relevancy between subscriptions and the message. The second one is location-aware AdSense, which extends traditional AdSense (www.google.com/adsense) to support location-aware services. The advertisers register their location-based advertisements (e.g., "`seafood`" at `Manhattan`) in the system. The system pushes relevant advertisements to mobile users based on their locations and contents they are browsing (e.g., webpages). The third one is tweet delivery. To receive feedback of their products in a specific area from Twitter, market analysts register their interests (e.g., "`ipad2`" at `LA`). For each tweet (e.g., "`ipad2 is expensive`" at `LA Airport`), the system pushes the tweet to relevant analysts whose spatio-textual subscriptions match the tweet.

One big challenge in a publish/subscribe system is to achieve high performance. A publish/subscribe system should support tens of millions of subscribers and deliver messages to relevant subscribers in milliseconds. Since messages and subscriptions contain both location information and textual description, it is rather costly to deliver messages to relevant subscribers. This calls for an efficient filtering technique to support location-aware publish/subscribe services.

To address the challenge, we propose a token-based R-tree index structure (called R^t-tree) by integrating each R-tree node with a set of tokens selected from subscriptions. Using the R^t-tree, we develop a filter-and-verification framework to efficiently deliver a message. To reduce the number of tokens associated with R^t-tree nodes, we select some *high-quality representative tokens* from subscriptions and associate them with R^t-tree nodes. This technique not only reduces index sizes but also improves the performance. Experiments on large, real data sets show that our method achieves high performance. We make the following

• *M. Yu, G. Li, T. Wang, and J. Feng are with the Department of Computer Science, Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China. E-mail: yumh12@mails.tsinghua.edu.cn, wangting421@gmail.com, {liguoliang, fengjh}@tsinghua.edu.cn.*
• *Z. Gong is with Department of Computer and Information Science, University of Macau, Macao 3974465, China. E-mail: fstzgg@umac.mo.*

Manuscript received 13 Feb. 2014; revised 24 July 2014; accepted 28 July 2014. Date of publication 0 . 0000; date of current version 0 . 0000. Recommended for acceptance by R. Jin.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2014.2349906

contributions: (1) We introduce a new computing model and formalize the location-aware publish/subscribe problem. (2) We propose a novel index structure, the R^t-tree, by integrating high-quality representative tokens selected from subscriptions into the R-tree nodes. Our method can support both conjunctive queries and ranking queries. (3) Using our proposed indexes, we develop efficient filtering algorithms and effective pruning techniques to improve the performance. (4) We present how to support dynamic updates efficiently.

2 Preliminaries

2.1 Problem Formulation

In a location-aware publish/subscribe system, subscribers register subscriptions to capture their interests. A subscription s includes a textual description $s.T$ and spatial information $s.R$, denoted by $s = (T, R)$. The spatial information is used to capture a subscriber's most interested region. We use the well-known minimum bounding rectangle (MBR) to denote a region $s.R$. The textual description is used to capture a subscriber's content-based interests, denoted by a set of tokens $s.T = \{t_1, t_2, \ldots, t_{|s.T|}\}$.

A message m posted by a publisher also contains a textual description $m.T$ and spatial information $m.R$, denoted by $m = (T, R)$, which respectively have the same meaning as those of subscriptions. Note that the spatial information $m.R$ of a message can be a point, e.g., mobile user's location. If the spatial information of a message is a point, we call it *point message*; otherwise we call it *range message*.

Let $\mathcal{S} = \{s_1, s_2, \ldots, s_{|\mathcal{S}|}\}$ denote the set of subscriptions. Given a subscription $s_i \in \mathcal{S}$ and a message m, a location-aware publish/subscribe system delivers m to s_i (s_i is called an answer of m), if they satisfy

1) *Spatial Constraint*. Message m and subscription s_i have spatial overlap (i.e., $s_i.R \cap m.R \neq \phi$) and;
2) *Textual Constraint*. All tokens in subscription s_i are contained in message m (i.e., $s_i.T \subseteq m.T$).

We first consider the conjunctive semantics for the textual constraints, that is any token in a subscription needs to be contained in the message. Our method can also support disjunctive semantics by decomposing a subscription to several small subscriptions. For example, we can decompose a subscription with tokens "(`iphone4s` or `ipad2`) and `AT&T`" to two subscriptions with tokens "`iphone4s` and `AT&T`" and "`ipad2` and `AT&T`". For the spatial constraint, we consider the case that a message and a subscription have spatial overlap. We will discuss how to support ranking semantics which finds subscriptions with similarity to the message larger than a given threshold by considering both textual relevancy and spatial proximity in Section 6. Based on these notations, we formalize the location-aware publish/subscribe problem as below.

Definition 1 (Location-aware Publish/Subscribe). *Given a set of subscriptions $\mathcal{S}$ and a message m, a location-aware publish/subscribe system delivers m to $s_i \in \mathcal{S}$ if $s_i.R \cap m.R \neq \phi$ and $s_i.T \subseteq m.T$.*

Example 1. Consider the 12 subscriptions and 2 messages in Fig. 1. For point message $m_p = (\{$`iphone4s`,`ipad2`, `AT&T`, `64GB`$\}, P_m)$, subscription $s_{12} = (\{$`64GB`, `ipad2`$\}, R_{12})$ is an answer. $s_{10} = (\{$`32GB`, `AT&T`, `iphone4s`$\}, R_{10})$ is not an answer as it has a token "`32GB`" which does not appear in m_p. $s_7 = (\{$`AT&T`, `ipad2`$\}, R_7)$ is not an answer as it has no spatial overlap with m_p. The answers of m_p are s_{11} and s_{12}. For a range message $m_r = (\{$`iphone4s`, `ipad2`, `AT&T`, `64GB`$\}, R_m)$, its answers are s_8, s_{11}, and s_{12}.

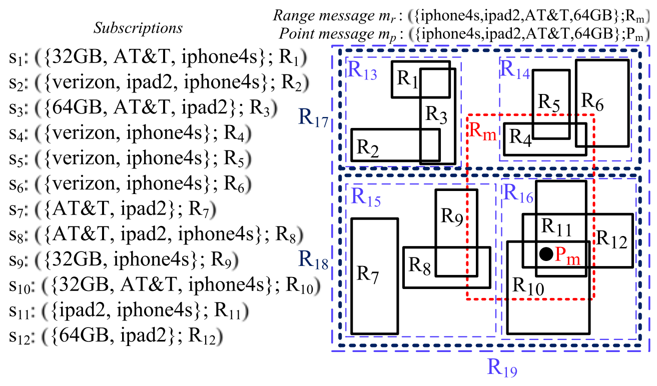

Subscriptions
Range message m_r: ({iphone4s,ipad2,AT&T,64GB};R$_m$)
Point message m_p: ({iphone4s,ipad2,AT&T,64GB};P$_m$)
s$_1$: ({32GB, AT&T, iphone4s}; R$_1$)
s$_2$: ({verizon, ipad2, iphone4s}; R$_2$)
s$_3$: ({64GB, AT&T, ipad2}; R$_3$)
s$_4$: ({verizon, iphone4s}; R$_4$)
s$_5$: ({verizon, iphone4s}; R$_5$)
s$_6$: ({verizon, iphone4s}; R$_6$)
s$_7$: ({AT&T, ipad2}; R$_7$)
s$_8$: ({AT&T, ipad2, iphone4s}; R$_8$)
s$_9$: ({32GB, iphone4s}; R$_9$)
s$_{10}$: ({32GB, AT&T, iphone4s}; R$_{10}$)
s$_{11}$: ({ipad2, iphone4s}; R$_{11}$)
s$_{12}$: ({64GB, ipad2}; R$_{12}$)
R$_{17}$
R$_{18}$
R$_{19}$
R$_{13}$
R$_{14}$
R$_{15}$
R$_{16}$
R$_1$
R$_2$
R$_3$
R$_4$
R$_5$
R$_6$
R$_7$
R$_8$
R$_9$
R$_{10}$
R$_{11}$
R$_{12}$
R$_m$
P$_m$

Fig. 1. An example of subscriptions and messages.

2.2 Related Work

There are some studies on location-aware publish/subscribe from a network perspective (e.g., routing, mobile networks) [6], [9], [10], [12], [13], [17], [19]. Different from these works, we focus on devising efficient algorithms to improve the performance in a centralized setting from a database perspective.

Spatial Keyword Search. There are many studies on spatial keyword search [4], [5], [7], [8], [15], [16], [21], [23], [25], [26], [29], [32], [33], [34], [35], [36]. The first problem is knn based keyword search, which, given a location and a set of keywords, finds top-k nearest neighbors by considering the distance and textual relevancy. Felipe et al. [16] integrated signature files and R-tree. Cong et al. [8] combined inverted files and R-tree. Cong et al. [4] studied how to find top-k prestige-based spatial objects. The second problem is region based keyword search, which, given a region and a keyword query, finds the relevant objects in this region. Zhou et al. [36] discussed several strategies to combine R-tree and inverted indexes. Hariharan et al. [21] integrated inverted lists into R-tree nodes. Chen et al. [7] extended this problem to support large numbers of "footprint representations." The third problem is collective keyword search, which, given a set of keywords, finds a set of close objects that match the keywords. Zhang et al. [33], [34] integrated keyword bitmap and keyword MBR into R-tree nodes to find the closest objects. Cao et al. [5] proposed several exact and approximate algorithms.

Yao et al. [32] studied the problem of approximate string match in spatial databases. Wu et al. [29] tackled the problem of spatial keyword search for moving objects. Lu et al. [25] extended reversed knn techniques to support reverse spatial and textual knn search. Roy and Chakrabarti [26] proposed materialization-based techniques to support type-ahead search. Leung et al. [22] used locations to improve personalized search.

The above problems substantially differ from our location-aware publish/subscribe problem, since they use a

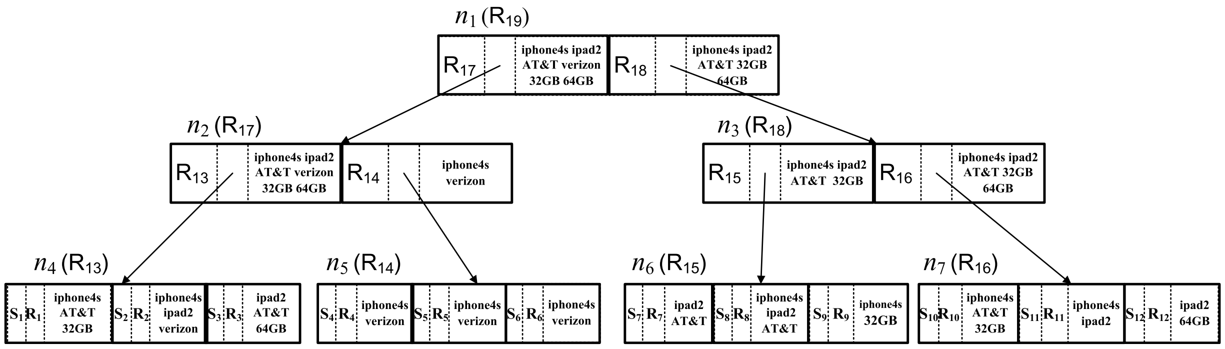

n_1 (R_{19})
R_{17} iphone4s ipad2 AT&T verizon 32GB 64GB
R_{18} iphone4s ipad2 AT&T 32GB 64GB
n_2 (R_{17})
R_{13} iphone4s ipad2 AT&T verizon 32GB 64GB
R_{14} iphone4s verizon
n_3 (R_{18})
R_{15} iphone4s ipad2 AT&T 32GB
R_{16} iphone4s ipad2 AT&T 32GB 64GB
n_4 (R_{13})
S_1 R_1 iphone4s AT&T 32GB
S_2 R_2 iphone4s ipad2 verizon
S_3 R_3 ipad2 AT&T 64GB
n_5 (R_{14})
S_4 R_4 iphone4s verizon
S_5 R_5 iphone4s verizon
S_6 R_6 iphone4s verizon
n_6 (R_{15})
S_7 R_7 ipad2 AT&T
S_8 R_8 iphone4s ipad2 AT&T
S_9 R_9 iphone4s 32GB
n_7 (R_{16})
S_{10} R_{10} iphone4s AT&T 32GB
S_{11} R_{11} iphone4s ipad2
S_{12} R_{12} ipad2 64GB

Fig. 2. R ^{t}-tree index for subscriptions in Fig. 1.

pull model and we employ a push model. Compared with our previous work [24], we make the following significant additions. First, we extended our conjunctive semantics to support ranking semantics. Second, we added how to support updates. Third, we analyzed the time complexity and space complexity.

Publish/Subscribe Services. Foltz and Dumais [18] studied the information filtering problem from the IR perspective. They used a vector space model and Latent Semantic Indexing (LSI) to achieve high effectiveness. Fabret et al. [14] and Aguilera et al. [1] studied the publish/subscribe problem from the database perspective. However their subscriptions are predicates which are different from ours. Yan and Garcia-Molina studied keyword-based filtering based on a binary model [31] and a vector space model [30]. There are many studies on XML filtering [2], [11]. Existing publish/subscribe methods consider textual descriptions while we consider both textual and spatial information.

Data Stream Management. There are many studies on data stream management [3], [20], [28], using a push model. Different from their problems, we focus on location-aware filtering for streaming messages.

2.3 Baseline Methods

We can extend existing techniques to support our problem and introduce several baseline methods.

Keyword-first method. It first uses existing contentbased publish/subscribe techniques to generate the candidates that satisfy the textual constraint, e.g, using inverted lists [31]. Then it verifies candidates to check whether they satisfy spatial constraint. Obviously this method generates large numbers of candidates and leads to low performance.

Spatial-first method. It first generates the locationbased candidates that satisfy the spatial constraint, using existing methods, e.g., segment tree or R-tree [27]. Then it filters candidates which do not satisfy textual constraint. This method also generates huge numbers of candidates and has poor performance.

Spatial keyword search based method. We can extend existing spatial keyword search methods, e.g., IRTree [8], to support our model. We construct an IRTree and traverse the IRTree from the root to leaves. For each node, we use the MBR to check the spatial constraint and utilize the inverted index to examine the textual constraint. However the IRTree has to visit many nodes unnecessarily and leads to low efficiency since if a node contains a token and satisfies the spatial constraint, it must access such node. To address this problem, we propose new indexes and filtering algorithms. In this paper, we focus on the in-memory setting to achieve instant performance.

3 R^t-TREE BASED METHOD

We first propose an index structure in Section 3.1 and then devise efficient filtering algorithms in Section 3.2.

3.1 R^t-TREE Index Structure

As the standard R-tree has no textual pruning power, we propose a token-based R-tree, called R^t-tree, by integrating tokens of subscriptions into R-tree nodes. R^t-tree is a balanced search tree. Each leaf node contains between b and B data entries, where each entry is a subscription. Each internal node has between b and B node entries. Each entry is a triple $\langle$Child, MBR, TokenSet$\rangle$, where Child is a pointer to its child node, MBR is the minimum bounding rectangle of all entries within this child, and TokenSet is a set of tokens selected from subscriptions (or a pointer to the token set). A leaf node's token set is the union of tokens of all subscriptions within this node and an internal node's token set is the union of token sets of all entries within this node. As an entry corresponds to a node, for simplicity a node is mentioned interchangeably with its corresponding entry if the context is clear.

Example 2. Fig. 2 shows an R^t-tree for subscriptions in Fig. 1. n_4, n_5, n_6, n_7 are leaf nodes. Each leaf node contains three subscriptions. For instance, n_4 contains three subscriptions s_1, s_2, s_3. n_2 has two entries (n_4, R_{13}, {`iphone4s`, `ipad2`, `AT&T`, `verizon`, `32GB`, `64GB`}) and (n_5, R_{14}, {`iphone4s`, `verizon`}). The first entry points to its child n_4 with MBR R_{13} and the token set is the union of the subscriptions of entries in its children, i.e., $s_1.T \cup s_2.T \cup s_3.T$. The second entry points to its child n_5 with MBR R_{14} and the token set is $s_4.T \cup s_5.T \cup s_6.T$.

Space Complexity. Suppose the height of the R^t-tree is $\mathcal{H}$ and the average number of tokens in subscriptions is $\mathcal{S}_{avg}$. Each token of a subscription is stored at most $\mathcal{H}$ times.[1] Thus the token-set complexity is $\mathcal{O}(\mathcal{H} \times \mathcal{S}_{avg} \times |\mathcal{S}|)$. There are at most $\frac{|\mathcal{S}|}{b} + \frac{|\mathcal{S}|}{b^2} + \cdots + \frac{|\mathcal{S}|}{b^{\mathcal{H}}} = |\mathcal{S}| \times \frac{\frac{1}{b} - \frac{1}{b^{\mathcal{H}+1}}}{1-\frac{1}{b}} \approx \frac{1}{b-1} \times |\mathcal{S}|$ nodes (Suppose the root also has b child nodes). The space complexity of

1. In this paper, we suppose the entries in the same internal node have no overlap, e.g., R^+-tree.

a node is $\mathcal{O}(B)$ for storing MBRs and child pointers. Thus the overall space complexity is

$$\mathcal{O}\left(\frac{B}{b-1} \times |\mathcal{S}| + \mathcal{H} \times \mathcal{S}_{avg} \times |\mathcal{S}|\right).$$

Updates. For the insertion of a new subscription s, we first locate the corresponding leaf node and insert the subscription into the leaf node as in the standard R-tree. The time complexity is $\mathcal{O}(\mathcal{H})$. Then we insert the tokens in the subscription to the TokenSets of the ancestors of the leaf node. The time complexity is $\mathcal{O}(\mathcal{H} \times |s|)$, where $|s|$ is number of tokens in subscription s. For the deletion operation of a subscription, we first remove the subscription from the corresponding leaf node similar to the standard R-tree operation with the time complexity $\mathcal{O}(\mathcal{H})$. Next we need to update the TokenSets of ancestors of the leaf node. To facilitate the update, we use a hash table to implement the TokenSet of a node. If a token is in a TokenSet, the hash value of the token is the number of subscriptions under the corresponding node of the TokenSet, which contain the token. Using the hash table, we can efficiently update TokenSets of ancestors of the corresponding leaf node of s as follows. Consider such a TokenSet. For each token in s, we decrease the hash value of the token by 1. If the hash value is 0, we remove the token from the hash table of the TokenSet. The time complexity of a deletion is $\mathcal{O}(\mathcal{H} \times |s|)$.[2]

3.2 Filtering Algorithms

We discuss how to use the R^t-tree to filter a message. We want to prune unnecessary nodes and only visit a small number of "pivotal" nodes, where a node is a *pivotal node* if there exist answers in its leaf descendants. Thus when traversing the R^t-tree, we only need to visit the *pivotal nodes*. However an R^t-tree node may have large numbers of leaf descendants and it is expensive to check whether a node is a pivotal node. Based on this observation, we propose a filter-and-verification framework. In the filter step, we find a set of *candidate nodes* which is a superset of pivotal nodes. In the verification step we verify the subscriptions in the *leaf candidate nodes* generated in the filter step. Next we introduce two filters. Given a node n, let $n.R$ denote its MBR and $n.T$ denote its token set which can be obtained from the corresponding entry in its parent node. We prune node n, if it satisfies

1) *MBR Filter*: $n.R \cap m.R = \phi$. It invalidates spatial constraint as subscriptions under node n (in n's leaf descendants) have no overlap with m; or
2) *Token Filter*: $n.T \cap m.T = \phi$. It invalidates the textual constraint. The reason is that any subscription under node n must contain a token in $n.T$ which does not appear in $m.T$, thus the subscription does not satisfy the textual constraint.

The nodes that are not pruned by the MBR filter and token filter are called *candidate nodes*. The subscriptions in the leaf candidate nodes are *candidate answers* of message m. Based on the two filters, we devise a filter-and-verification algorithm.

2. We can update token sets in a delay manner for deletions, and our method can still correctly and completely find the answers (see Section 3.2).

```
Algorithm 1: R^t-TREE ($\mathcal{S}, m$)
  Input: $\mathcal{S}$: A subscription set; $m$: A message
  Output: $\mathcal{R}$: Answers of $m$
1 Build an R$^t$-tree with root $r$ ;
2 Initialize a candidate node set $\mathcal{CN} = \phi$ ;
3 R$^t$-TREE-FILTER ($r$, $m$, $\mathcal{CN}$) ;
4 $\mathcal{R}$ = R$^t$-TREE-VERIFY ($m$, $\mathcal{CN}$) ;

Function R$^t$-TREE-FILTER ($r$, $m$, $\mathcal{CN}$)
  Input: $r$: A tree node; $m$: A message
         $\mathcal{CN}$: A set of leaf candidate nodes
  Output: $\mathcal{CN}$: updated candidate-node set
1 if $r$ is a leaf node then $\mathcal{CN} \leftarrow r$ ;
2 else
3     for each entry $n$ in $r$ do
4         pruned = MBRTokenFilter ($n, m$);
5         if not pruned then R$^t$-TREE-FILTER
          ($n$,$m$,$\mathcal{CN}$);

Function R$^t$-TREE-VERIFY ($m$, $\mathcal{CN}$)
  Input: $m$: A message;
         $\mathcal{CN}$: A set of leaf candidate nodes
  Output: $\mathcal{R}$: Answers of $m$
1 for each entry $s \in \mathcal{CN}$ do
2     if $s.R \cap m.R \neq \phi$ & $s.T \subseteq m.T$ then $\mathcal{R} \leftarrow s$ ;
```

Fig. 3. R ^{t}-TREE-based algorithm.

Given a message m, we traverse the R^t-tree in pre-order. From the root node, we scan each of its entries, e.g., node n. If node n satisfies one of the two filters, i.e., $n.R \cap m.R = \phi$ or $n.T \cap m.T = \phi$, we prune node n; otherwise we visit n's children and repeat the above steps. Iteratively, we can find all leaf candidate nodes. For each subscription s on leaf candidate nodes, we check whether $s.R \cap m.R \neq \phi$ and $s.T \subseteq m.T$. If yes, s is an answer. Fig. 3 illustrates the R^t-tree based algorithm. It first constructs an R^t-tree (line 1) and calls function R^t-TREE-FILTER to find candidate nodes (line 3). Then it uses function R^t-TREE-VERIFY to verify the candidates (line 4). R^t-TREE-FILTER calls function MBRTokenFilter to check whether n satisfies the MBR filter and token filter, which will be discussed later. If node n does not satisfy the two filters, n is a candidate node and R^t-TREE-FILTER examines n's child nodes (line 5).

Example 3. Consider the R^t-treein Fig. 2 and a message $m = (\{\texttt{ipad2}, \texttt{AT\&T}, \texttt{32GB}, \texttt{64GB}\}, \texttt{R}_\texttt{m})$, where R_m is the dashed MBR in Fig. 1. The root node has two entries: node n_2 and node n_3. Node n_2 is not pruned by the MBR filter as it has overlap with $m.R$. n_2 is not pruned by the token filter as it contains "ipad2" which appears in message m. Thus node n_2 is a candidate node. Node n_2 has two entries: n_4 and n_5. As node n_4 has no overlap with $m.R$, it is pruned by the MBR filter. Node n_5 has spatial overlap with $m.R$ and cannot be pruned by the MBR filter. As n_5's token set {iphone4s, verizon} has no common token with $m.T$, n_5 is pruned by the token filter. For node n_3, we access its first entry n_6. As n_6 is not pruned by the MBR filter and token filter, it is a leaf candidate node. We verify subscriptions s_7, s_8, s_9 in n_6. We prune s_7 as it has no spatial overlap with $m.R$. We prune s_8 and s_9 as they have a token "iphone4s" which does not appear in $m.T$. Similarly n_7 is also a leaf candidate node and we verify s_{10}, s_{11}, s_{12} in n_7. We prune s_{10}

and s_{11} as their tokens are not contained in $m.T$ and get an answer s_{12}.

MBRTokenFilter. We discuss how to efficiently implement the two filters. To check whether $n.R \cap m.R = \phi$, we examine whether $n.R$ has a vertex contained in $m.R$ and the time complexity is $\mathcal{O}(1)$. To check whether $n.T \cap m.T = \phi$, we use the hash table of n's TokenSet to do the checking as follows. For each token in $m.T$, if it is contained in the hash table, $n.T \cap m.T \neq \phi$ and we terminate; otherwise we check the next token in $m.T$. Iteratively we can implement the token filter and the time complexity is $\mathcal{O}(|m.T|)$.

Verification. To verify a subscription s, we need to check whether $s.R \cap m.R \neq \phi$ and $s.T \subseteq m.T$. The time complexity of checking $s.R \cap m.R \neq \phi$ is $\mathcal{O}(1)$. To check $s.T \subseteq m.T$, we first sort the tokens in $m.T$ with complexity $\mathcal{O}(|m.T| \times \log|m.T|)$ (We will discuss different sorting strategies in Section 4.2). Then we use each token in $s.T$ to do a binary search in $m.T$ and the complexity is $\mathcal{O}(|s.T| \times \log|m.T|)$.

Time Complexity. Let $\mathcal{C}_{R^t}$ denote the set of candidate nodes in the R^t-tree based algorithm. For each candidate node c, we check whether each of its entry (n) satisfies the MBR filter and token filter. The time complexity of the filter step is

$$\mathcal{O}\left(\sum_{c\in\mathcal{C}_{R^t}}\sum_{n\in c}|m.T|\right). \quad (1)$$

Let $\mathcal{C}^l_{R^t}$ denote the set of leaf candidate nodes. In the verification step, for each subscription s on each leaf candidate node l, we examine whether $s.R \cap m.R \neq \phi$ and $s.T \subseteq m.T$. As we only need to sort $m.T$ once, the time complexity of the verification step is

$$\mathcal{O}\left(|m.T| \times \log|m.T| + \sum_{l\in\mathcal{C}^l_{R^t}}\sum_{s\in l}(|s.T| \times \log|m.T|)\right). \quad (2)$$

Notice that an algorithm should satisfy (1) Completeness: any subscription satisfying the spatial constraint and textual constraint must be found by the algorithm; and (2) Correctness: any subscription found by the algorithm must satisfy the two constraints. We prove that the R^t-tree based algorithm satisfies the two properties as stated in Theorem 1.

Theorem 1. *The R^t-tree based algorithm satisfies completeness and correctness.*

Proof. The correctness is obvious as we use a verification step to verify candidates. Thus we only need to prove completeness. Consider a subscription s which satisfies the textual constraint and spatial constraint. Suppose s is in leaf node l. As s is an answer, l's ancestors must satisfy the MBR constraint. As s's tokens are added into l's ancestors' token sets, its ancestors must satisfy the textual constraint. Thus node l must be a leaf candidate node, and s must be taken as an answer. Thus the theorem holds. □

4 Single Representative Tokens

The R^t-tree has to maintain many tokens in each node and leads to low pruning power. To address this issue, in this section, we propose an effective technique to reduce the number of tokens in each node which not only reduces index sizes but improves performance.

4.1 Representative Tokens

Different from TokenSets on R^t-tree nodes, we select high-quality *representative tokens* and use *representative-token sets* to replace TokenSets. To differentiate this method from R^t-tree, we call it R^{t+}-tree. Next we discuss how to select representative tokens. For each subscription, we select a single token as its *representative token.* For each leaf, its representative-token set is the set of representative tokens of subscriptions in this node. For each internal node, its representative-token set is the union of representative-token sets of its children. We have an observation that if the representative-token set of a node has no common token with a message, we can prune the node as stated in Lemma 1. This is because any subscription under this node cannot satisfy textual constraint as it contains tokens which are not in the message.

Lemma 1. *Given an R^{t+}-tree node n and a message m, if n's representative-token set has no common token with $m.T$, any subscription under node n cannot be an answer of m.*

Proof. Consider any subscription s under node n. s must contain a token in n's representative-token set. As n's representative-token set has no common token with $m.T$, s must contain a token which is not in $m.T$. Thus s does not satisfy the textual constraint and is not an answer. Thus any subscription under node n cannot be an answer. □

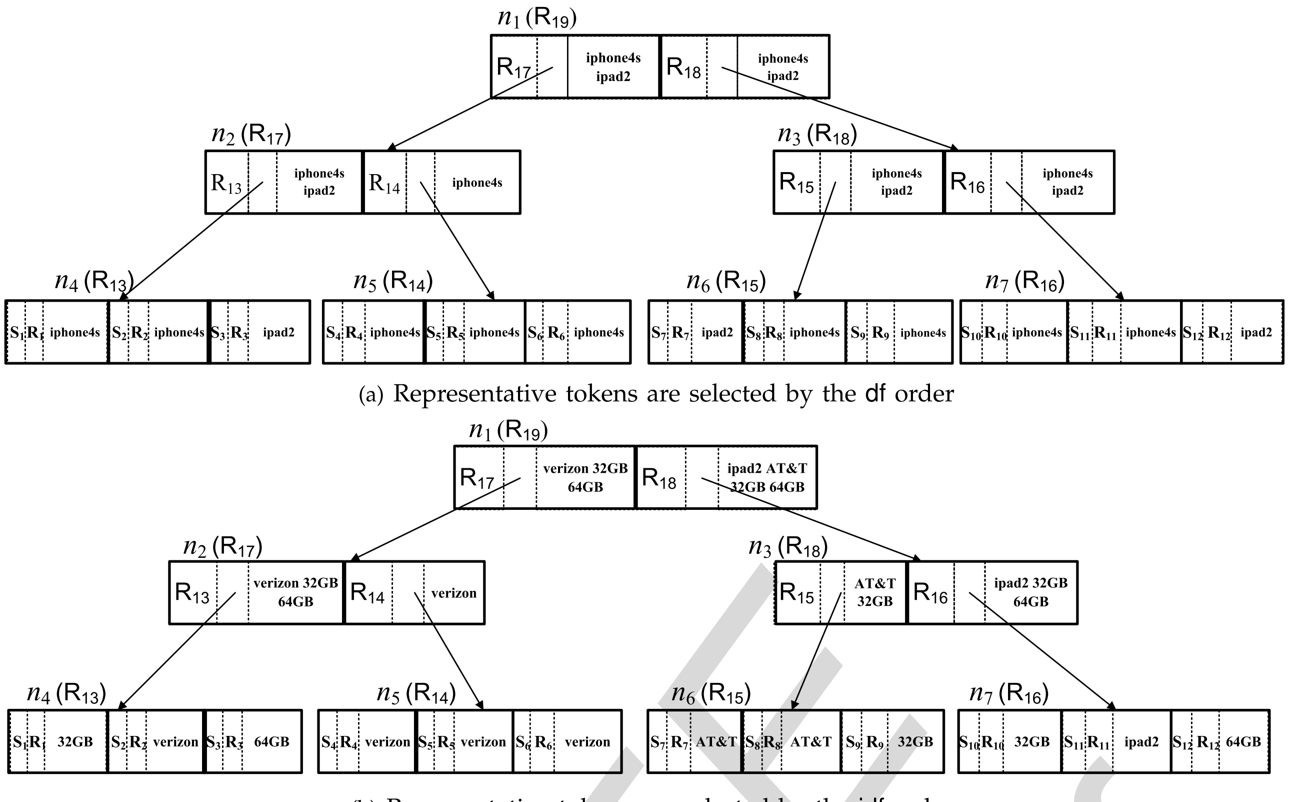
Example 4. Consider the subscriptions in Fig. 1. We construct an R^{t+}-tree as shown in Fig. 4a. For subscriptions s_1, s_2, s_3, we respectively select representative tokens "iphone4s", "iphone4s", "ipad2". Thus the representative-token set of node n_4 is {iphone4s, ipad2}. For s_4, s_5, s_6, we select representative tokens "iphone4s". Thus the representative-token set of node n_5 is {iphone4s}. The representative-token set of n_2 is {iphone4s, ipad2} which is the union of the representative tokens of its child nodes n_4 and n_5. For a message $m = (\{\texttt{AT\&T}, \texttt{32GB}, \texttt{64GB}\}, \texttt{R}_\texttt{m})$, R^t-tree cannot prune node n_2 as its token set shares a token "AT&T" with m(Fig. 2). Instead R^{t+}-tree can prune n_2 as its representative-token set shares no common token with m.

Space Complexity. There are at most $|\mathcal{S}|$ representative tokens and each representative token is stored at most $\mathcal{H}$ times. Thus the complexity of representative-token sets is $\mathcal{O}(\mathcal{H} \times |\mathcal{S}|)$. The R^{t+}-tree has the same MBR size with the R^t-tree. Thus the space complexity of the R^{t+}-tree is

$$\mathcal{O}\left(\frac{B}{b-1} \times |\mathcal{S}| + \mathcal{H} \times |\mathcal{S}|\right).$$

Updates. The insertion (deletion) is similar to that on the R^t-tree, except we only need to insert (delete) the representative token into (from) the representative-token sets of ancestors of the corresponding leaf node of a message. The time complexity is $\mathcal{O}(\mathcal{H})$.

We still adopt the filter-and-verification framework in Section 3.2 to filter a message. Theorem 2 shows correctness and completeness.



(a) Representative tokens are selected by the df order

(b) Representative tokens are selected by the idf order

Fig. 4. R ^{t}-tree with representative tokens (df of tokens: `iphone4s`:9, `ipad2`:6, `AT&T`:5, `verizon`:4, `32GB`:3, `64GB`:2).

Theorem 2. *The R^{t+}-tree based algorithm satisfies completeness and correctness.*

Proof. This theorem can be proved based on Theorem 1 and Lemma 1. □

Comparison of R^t-tree and R^{t+}-tree. The R^{t+}-tree involves smaller index sizes than the R^t-tree. More interestingly, we prove the R^{t+}-tree has larger pruning power than the R^t-tree: if the R^t-tree prunes a node, the R^{t+}-tree also prunes the node, as stated in Lemma 2. R^{t+}-tree is specially designed for conjunctive queries but not effective for disjunctive queries.

Lemma 2. *Let $\mathcal{C}_{R^t}$ and $\mathcal{C}_{R^{t+}}$ respectively denote the candidate node sets of the R^t-tree based method and the R^{t+}-tree based method, we have $\mathcal{C}_{R^t} \supseteq \mathcal{C}_{R^{t+}}$.*

Proof. For any node n pruned by the R^t-tree, we only need to prove that it must be pruned by the R^{t+}-tree. If n is pruned by MBR filter in the R^t-tree, then it is also pruned in the R^{t+}-tree. If n is pruned by token filter in the R^t-tree, then the token set of node n has no common token with $m.T$. As the representative-token set is a subset of the token set, it also has no overlap with $m.T$, thus the R^{t+}-tree must prune node n and the Lemma is proved. □

Example 5. Recall the R^t-tree in Fig. 2. Consider a message m=({AT&T, 32GB, 64GB}, $\mathtt{R_m}$). Node n_3 shares common tokens with m, thus the R^t-tree cannot prune node n_3. However all subscriptions under n_3 contain tokens "iphone4s" or "ipad2" which do not appear in m. Thus n_3 can be pruned. In the R^{t+}-tree, the representative-token set of n_3 is {iphone4s, ipad2} which has no common token with m, thus the R^{t+}-tree can prune n_3.

4.2 Selecting Representative Tokens

There are multiple ways to select representative tokens from subscriptions. We introduce several methods to select high-quality representative tokens.

Random selection. A naive method is to randomly select a representative token for each subscription. This method does not utilize the token distribution and can be optimized.

The df-based method. Based on time complexity in Equation 1, to improve the filtering performance, we should reduce the number of candidate nodes. Intuitively the smaller sizes of representative-token sets, the larger pruning power, and the smaller number of candidates. To achieve high performance, it is important to reduce the sizes of the representative-token sets. However we find the problem of minimizing the representative-token set is an NP-hard problem which can be proved by a reduction from the minimum hitting set problem or the set cover problem.

To address this issue, we propose a greedy algorithm. We select representative tokens based on the df order as follows. We first select the token with the largest df and take it as the representative token of subscriptions that contain the token. Then we remove all such subscriptions and compute the frequencies of each token in the remainder subscriptions. Next we select the token with the largest df in the remainder subscriptions. We terminate the algorithm if there is no subscription left. Iteratively we can generate the representative token for each subscription.

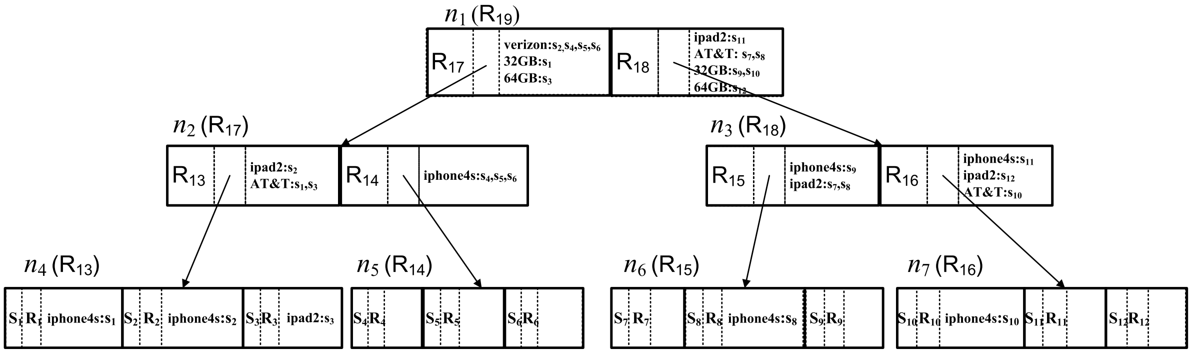


Fig. 5. An R^t-tree with multiple representative tokens (`iphone4s`:9, `ipad2`:6, `AT&T`:5, `verizon`:4, `32GB`:3, `64GB`:2).

Example 6. Fig. 4a shows the R^{t+}-tree with representative tokens selected by the df order. As "`iphone4s`" has the largest df (its df is 9) and it appears in $s_1, s_2, s_4-s_6, s_8-s_{11}$, we select "`iphone4s`" as their representative tokens. Then in the remainder subscriptions, i.e., s_3, s_7, s_{12}, "`ipad2`" has the largest df, we select "`ipad2`" as their representative tokens. For node n_4, the subscriptions under n_4 are s_1, s_2, s_3, thus its representative-token set is {`iphone4s`, `ipad2`}. On the R^{t+}-tree there are 23 representative tokens and on the R^t-tree there are 59 tokens. Obviously the df order can reduce token-set sizes.

The idf-based method. A token with higher frequency usually has larger probability appearing in a message. If we add such a token into the token set, the token set has larger probability sharing tokens with the message, thus the token set has lower pruning power. As the df order selects the representative tokens with the highest frequencies, the df based method may lead to low pruning power. To address this issue, for each subscription, we select the token with the largest idf as its representative token.

Example 7. Fig. 4b shows the R^{t+}-tree with representative tokens selected by the idf order. We respectively select "`32GB`, `verizon`, `64GB`" for subscriptions s_1, s_2, s_3 with the largest idf. The representative-token set of node n_2 is the union of these three representative tokens. Consider message $m = (\{\texttt{iphone4s}, \texttt{ipad2}, \texttt{AT\&T}, \texttt{64GB}\}, \texttt{R}_\texttt{m})$. The df order cannot prune node n_5 as its representative-token set {`iphone4s`} shares a common token with message m (Fig. 4a). However the idf order can prune node n_5 as its representative-token set {`verizon`} shares no common token with message m (Fig. 4b).

Locality-aware method. Tokens usually have different frequencies in different locations. For example, in Fig. 1 token "`verizon`" has a large frequency in node $n_5(R_{14})$ and low frequency in nodes $n_6(R_{15})$ and $n_7(R_{16})$. We should consider the locality-aware token distribution to count token frequencies. To address this issue, we partition the whole region into several grids and count the frequencies of tokens for each grid. Thus we can capture the locality-aware frequencies. Then for each subscription, we first sort its tokens based on by their frequencies in the grid where the subscription locates in and then use the df-based method or the idf-based method to select representative tokens for the subscription.

Updates. To get the df/idf order, we use a hash table to maintain the frequency of each token. For the insertion of a new subscription, we first get the frequency of each token using the hash table and sort the frequency to get the df/idf order. Then we select a representative token using the above methods and insert the subscription and the representative token into the R^{t+}-tree as discussed in Section 4.1. Next we update the frequency of each token. We do not update the representative tokens of original subscriptions. For the locality-aware method, we can maintain a hash table for each grid to support efficient updates.

5 Multiple Representative Tokens

Since the R^{t+}-tree only selects a single representative token, it does not take full advantage of the textual information to do textual pruning. To address this issue, we propose selecting multiple representative tokens and devise an efficient filtering algorithm to directly find answers without the verification step.

5.1 Multiple Representative Tokens

Like the R^{t+}-tree, we also associate each node with a representative-token set. Different from the R^{t+}-tree, given a subscription s, we do not repeatedly use a single token for all ancestors of the corresponding leaf node. Instead we select different representative tokens for different ancestors and each ancestor contains a token of s. If s has larger than $\mathcal{H}$ tokens, we insert its last $|s| - \mathcal{H} + 1$ tokens into the leaf. If s has smaller than $\mathcal{H}$ tokens, only the ancestors in the first $|s|$ levels contain a token. Formally, let $s.T[i]$ denote the ith token in s and we assign $s.T[i]$ to its ancestor at the ith level. For each node n on the ith level, its representative-token set is the set of ith tokens of subscriptions under node n, i.e., $\cup_{s \in \mathcal{S}_n} s.T[i]$, where $\mathcal{S}_n$ is the set of subscriptions under node n. Let $n.T$ denote the representative-token set of node n. For each token $t \in n.T$, we build an inverted list $\mathcal{I}_n[t]$, which is the set of subscriptions in $\mathcal{S}_n$ whose ith token is t, i.e., $\mathcal{I}_n[t] = \{s \mid s \in \mathcal{S}_n \text{ and } s.T[i] = t\}$.

Example 8. Fig. 5 shows the R^{t++}-tree for subscriptions in Fig. 1. For s_1, $s_1.T[1] =$ "`32GB`", $s_1.T[2] =$ "`AT&T`", $s_1.T[3] =$ "`iphone4s`". Consider the first entry at the first level (the root node), i.e., node n_2. Node n_2 contains six subscriptions $s_1, s_2, \ldots, s_6$, thus the subscription set under node n_2 is $\mathcal{S}_{n_2} = \{s_1, \cdots s_6\}$. $s_1.T[1] =$ "`32GB`", $s_2.T[1] =$ "`verizon`", $\texttt{s}_3.\texttt{T}[1] =$ "`64GB`", $\texttt{s}_4.\texttt{T}[1] =$ "`verizon`", $s_5.T[1] =$ "`verizon`", and $s_6.T[1] =$ "`verizon`". The representative-token set of node n_2 is $n_2.T = s_1.T[1] \cup s_2.T[1] \cup \cdots \cup s_6.T[1] =$ {`verizon`, `32GB`, `64GB`}. The inverted list of "verizon" in node n_2 is $\mathcal{I}_{n_2}[\texttt{verizon}] = \{\mathtt{s_2, s_4, s_5, s_6}\}$. Similarly for node n_5, it contains three subscriptions, s_4, s_5, s_6. The second tokens of these subscriptions are "iphone4s". Thus the token set of n_5 is {iphone4s} and the inverted list of iphone4s in node n_5 is $\{s_4, s_5, s_6\}$.

Property of The R^{t++}*-tree.* The R^{t++}-tree has a good property. For any subscription s, s appears exactly $|s|$ times on the inverted lists in the R^{t++}-tree. For any $i < |s|$, s appears exactly i times in the first i levels.

For example, consider subscription s_8 with tokens {AT&T, ipad2, iphone4s}. s_8 appears three times in the R^{t++}-tree, i.e., the inverted list of AT&T in node n_1 at level 1, the inverted list of ipad2 in node n_3 at level 2, and the inverted list of iphone4s in node n_6 at level 3. Obviously s_8 appears once in the first level and twice in the first two levels.

Space Complexity. The R^{t++}-tree has the same MBR size with the R^{t}-tree but has much smaller token-set size. We first analyze the token-set size. For each token of a subscription, it appears in exactly one R^{t++}-tree node. Thus the total size of representative-token sets is $\mathcal{O}(|\mathcal{S}_{avg}| \times |\mathcal{S}|)$. Then we consider inverted-list size. Each subscription s appears in exactly $|s.T|$ inverted lists, thus the total size of inverted lists is also $\mathcal{O}(|\mathcal{S}_{avg}| \times |\mathcal{S}|)$. Obviously the size of token sets and inverted lists is proportional to the data size. Plus the MBR sizes, the overall space complexity is

$$\mathcal{O}\left(\frac{B}{b-1} \times |\mathcal{S}| + \mathcal{S}_{avg} \times |\mathcal{S}|\right).$$

Updates. The insertion (deletion) of a subscription s is similar to that on the R^{t}-tree, except that we only need to insert (delete) the subscription into (from) the corresponding inverted lists of ancestors of the leaf node with respect to message s. We still use a hash table to implement the inverted list and thus the complexity to insert (remove) the subscription into (from) an inverted list is $\mathcal{O}(1)$. Hence the time complexity for an insertion (a deletion) is $\mathcal{O}(\mathcal{H} + |s|)$.

Discussion on token assignment. We have different ways to assign the tokens to different ancestors (Section 4.2). As the tokens with large idf usually have large pruning power, it is better to keep tokens with large idf in the upper-level nodes (see Section 7.1).

5.2 Filtering Algorithms

Consider a message m. We traverse the R^{t++}-tree from the root. For a node n in the ith level, if $n.R \cap m.R = \phi$, we prune node n as the message invalidates the spatial constraint; otherwise for each token $t \in n.T \cap m.T$, we retrieve the corresponding inverted list $\mathcal{I}_n[t]$. For each subscription $s \in \mathcal{I}_n[t]$, we count its occurrence number in the first i levels (i.e., the number of inverted lists of ancestors of node n that contain the subscription), denoted by cand_s. We will discuss how to compute cand_s later. Here based on cand_s, we have the following observations.

Case 1—$\mathsf{cand}_s < i$. In this case, subscription s appears smaller than i times in the first i levels. We can prove that s is not an answer as formalized in Lemma 3. The reason is as follows. As s appears in the ith level, it contains at least i tokens. For each node on the path from the root to node n, s must have a token in the node. If $\mathsf{cand}_s < i$, s must have a token which does not appear in m. Thus s invalidates the textual constraint and cannot be an answer.

Lemma 3. *Consider a message m and a node n at the i-th level. For any subscription s on the inverted lists of node n, if $\mathsf{cand}_s < i$, s cannot be an answer.*

Proof. We prove it by contradiction. Suppose s is an answer of m. As s appears in the ith level, its first i-tokens must be on the nodes from the root to node n. As s is an answer of m, these i tokens must be contained in m. Thus $\mathsf{cand}_s \geq i$ which contradicts with $\mathsf{cand}_s < i$. Therefore s cannot be answer. □

For example, consider $m = (\{\texttt{ipad2}, \texttt{64GB}\}, \mathtt{R_m})$ and node n_3 at the second level in Fig. 5. For subscription s_7, we have $\mathsf{cand}_{s_7} = 1$. We can deduce that s_7 cannot be an answer of the message. The main reason is as follows. As s_7 has 2 tokens, it must appear twice in the first two levels and its occurrence number should be two. However its occurrence number cand_{s_7} is 1, thus s_7 must contain a token which does not appear in m and s_7 is not an answer.

Case 2—$\mathsf{cand}_s = i$. In this case, we have two subcases.

Case 2.1—$\mathsf{cand}_s = i = |s|$. In this case, if $s.R \cap m.R \neq \phi$, s must be an answer of m as stated in Lemma 4. The basic idea is as follows. Each token of s appears in the R^{t++}-tree once. As $\mathsf{cand}_s = |s|$, all tokens in s are contained in m, thus s satisfies the textual constraint. As $s.R \cap m.R \neq \phi$, s satisfies the spatial constraint. Thus s must be an answer.

Lemma 4. *Consider a message m and a node n at the ith level. For any subscription s on the inverted lists of node n, if $\mathsf{cand}_s = |s|$ and $s.R \cap m.R \neq \phi$, s must be an answer.*

Proof. As s has overlap with $m.R$, we only need to prove $s.T \subseteq m.T$. As a subscription appears exactly $|s|$ times in the R^{t++}-tree and cand_s=$|s|$, each token of s must be contained in m. Thus s is an answer. □

For example, consider message $m_r = (\{\texttt{iphone4s}, \texttt{ipad2}, \texttt{AT\&T}, \texttt{64GB}\}, R_m)$ and node n_6 at the third level in Fig. 5. For subscription s_8, we have $\mathsf{cand}_{s_8} = 3$ and $s_8 \cap m_r.R \neq \phi$. We can deduce that s_8 must be an answer of m_r. The reason is as follows. First s_8 has three tokens and $\mathsf{cand}_{s_8} = 3$. That is all of its tokens must be contained in the message. Thus s_8 satisfies the token constraint. Second, as $s_8 \cap m_r.R \neq \phi$, s_8 satisfies the MBR constraint.

Case 2.2—$\mathsf{cand}_s = i < |s|$. In this case, if n is an internal node, there may exist answers under this node, and we need to visit n's children and repeat the above steps. On the contrary, if n is a leaf node, s cannot be an answer. This is because s only has $i < |s|$ tokens that appear in message m (i.e., it has at least one token which does not appear in m).

Consider $m = (\{\texttt{ipad2}, \texttt{AT\&T}, \texttt{64GB}\}, \mathtt{R_m})$ and node n_6 on the third level. For subscription s_8, we have $\mathsf{cand}_{s_8} = 1$. We can deduce s_7 cannot be an answer. The main reason is as follows. As s_7 has two tokens, it must appear twice in the first two levels and its occurrence number must be two. As its occurrence number cand_{s_7} is 1, it must contain a token which does not appear in m. Thus s_7 cannot be an answer.

Case 3—$\mathsf{cand}_s > i$. In this case n must be a leaf node since if n is an internal node, $\mathsf{cand}_s \leq i$ as each internal node contains at most one token of s. If $\mathsf{cand}_s=|s|$, it is similar to Case 2.1; If $\mathsf{cand}_s<|s|$, it is similar to Case 2.2.

Based on the above analysis, we devise an efficient algorithm to filter a message. We still traverse the R^{t++}-tree from the root in pre-order. Given a node n, for each token $t \in n.T \cap m.T$, we retrieve the corresponding inverted list $\mathcal{I}_n[t]$. For each subscription $s \in \mathcal{I}_n[t]$, we count its occurrence number cand_s. If $\mathsf{cand}_s = |s|$ and $s.R \cap m.R \neq \phi$, s is answer and added into the result set. If there exists a subscription s such that $\mathsf{cand}_s = i < |s|$ and n is not a leaf node, there may exist answers under node n. We access the node and repeat the above steps; otherwise if there has no such subscription, we prune node n. The main reason is as follows. First, all subscriptions with no smaller than i tokens under node n cannot be an answer of m (Lemma 5). Second, for all the subscriptions with smaller than i tokens, if they are answers, they must be added as results when accessing n's ancestors.

Lemma 5. *Consider a message m and a non-leaf node n at the ith level. If there does not exist a subscription s such that $\mathsf{cand}_s = i < |s|$, all subscriptions with no smaller than i tokens under node n cannot be an answer of m.*

Proof. For any subscription s with no smaller than i tokens, there are i tokens of s which are assigned with nodes in the first i levels. As s's occurrence number is smaller than i, thus there exists a token of s which does not appear in m. Hence s cannot be an answer. As the proof is true for all such subscriptions, the lemma is proved. □

Computing cand_s. To efficiently compute the occurrence number cand_s, we use a hash map $\mathcal{M}$ to maintain the occurrence numbers. Given a message m and a node n, if $n.R \cap m.R \neq \phi$, we use the hash-based method (Section 3.2) to compute $m.T \cap n.T$. For each token $t \in m.T \cap n.T$, we access its inverted list $\mathcal{I}_n[t]$. For each subscription s in $\mathcal{I}_n[t]$, we increase $\mathcal{M}[s]$ by 1. Notice that as s may be in multiple inverted lists on a leaf node, when computing its occurrence number, we consider all such lists. Obviously $\mathsf{cand}_s = \mathcal{M}[s]$.

The R^{t++}-tree based Algorithm. We devise an R^{t++}-tree based algorithm as illustrated in Fig. 6. R^{t++}-TREE first constructs an R^{t++}-tree with root r (line 1) and initializes a hash map $\mathcal{M}$ (line 2). Then it calls function R^{t++}-TREE-PRUNE to filter message m. R^{t++}-TREE-PRUNE first scans each entry (node n) from the root. If n does not satisfy spatial constraint (line 3), R^{t++}-TREE-PRUNE prunes the node (line 3); otherwise R^{t++}-TREE-PRUNE computes the intersection of its representative-token set and $m.T$ (line 4). Then for each token t in the intersection, R^{t++}-TREE-PRUNE accesses its inverted list $\mathcal{I}_n[t]$ and for each s on $\mathcal{I}_n[t]$, it increases $\mathcal{M}[s]$ by 1 to count its occurrence number (line 6). If $\mathcal{M}[s] = i < |s|$ and n is not a leaf node, R^{t++}-TREE-PRUNE visits n's children (To avoid repeatedly visiting n's children, we set visitFlag as true in line 8 and if visitFlag is true, we visit such children in line 12). If $\mathcal{M}[s] = |s|$ and $s.R \cap m.R \neq \phi$, s is an answer and added into the result (line 10).

Time Complexity. Let $\mathcal{C}_{R^{t++}}$ denote the set of nodes visited by the R^{t++}-tree based algorithm. For each entry n of a node, we compute the intersection of $n.T$ and $m.T$ using the hash-based method with time complexity $\mathcal{O}(|m.T|)$. We also need to access the inverted list of tokens in the intersection with complexity $\sum_{t\in n.T\cap m.T} |\mathcal{I}_n[t]|$. Thus the total time complexity is

$$\mathcal{O}\left(\sum_{c\in\mathcal{C}_{R^{t++}}} \sum_{n\in c} \left(|m.T| + \sum_{t\in n.T\cap m.T} |\mathcal{I}_n[t]|\right)\right).$$

```
Algorithm 2: R^{t++}-TREE ($\mathcal{S}, m$)
  Input: $\mathcal{S}$: A subscription set; $m$: A message
  Output: $\mathcal{R}$: Answers of $m$
1 Build an R^{t++}-tree with root $r$;
2 Initialize a hash map $\mathcal{M}$ ;
3 R^{t++}-TREE-PRUNE ($r, m, \mathcal{R}, \mathcal{M}$) ;

Function R^{t++}-TREE-PRUNE ($r, m, \mathcal{R}, \mathcal{M}$)
  Input: $r$: An R^{t++}-tree node; $m$: A message
         $\mathcal{R}$: Answers of $m$; $\mathcal{M}$: Hash map
  Output: $\mathcal{R}$: Answers of $m$
1  visitFlag = false;
2  for each entry $n$ in node $r$ do
3      if $n.R \cap m.R = \phi$ then return;
4      for token $t \in n.T \cap m.T$ do
5          for each subscription $s$ in $\mathcal{I}_n[t]$ do
6              $\mathcal{M}[s] = \mathcal{M}[s] + 1$;
7              if !visitFlag & $\mathcal{M}[s] = i < |s|$ & $n$ is not a leaf node then
8                  visitFlag = true;
9              if $\mathcal{M}[s] = |s|$ & $s.R \cap m.R \neq \phi$ then
10                 $\mathcal{R} \leftarrow s$ ;
11     if visitFlag then
12         R^{t++}-TREE-PRUNE ($n, m, \mathcal{R}, \mathcal{M}$);
```

Fig. 6. R^{t++}-TREE based algorithm.

The R^{t++}-tree based algorithm satisfies completeness and correctness as stated in Theorem 3.

Theorem 3. *The R^{t++}-tree based algorithm satisfies completeness and correctness.*

Proof. Completeness: Suppose a subscription s on leaf node l is an answer of message m. As s is an answer of m, it will appear on l's ancestor nodes from the root to node l. As it satisfies the spatial constraint, any of its ancestors also satisfies the spatial constraint. As any token of s is contained by m, our algorithm will find the subscription in each level. Thus its occurrence number will be $|s|$, and s must be found as an answer.

Correctness: We can prove it based on Lemma 4. □

Comparition of R^{t}-tree,R^{t+}-tree, R^{t++}-tree. R^{t+}-tree has the smallest index size but it is only effective for conjunctive queries. R^{t++}-tree achieves the highest performance. Since R^{t}-tree does not need to maintain the token order, R^{t}-tree is much simpler than R^{t+}-tree and R^{t++}-tree and has the lowest update cost.

6 EXTENDING TO RANKING SEMANTICS

We extend our techniques to support ranking semantics. Given a subscription $s \in \mathcal{S}$ and a message m, we evaluate their similarity (relevancy) as below.

$$\psi(s,m) = \alpha\psi_s(s,m) + (1-\alpha)\psi_t(s,m), \tag{3}$$

where α is a tuning parameter, ψ_s is a spatial similarity function and ψ_t is a textual similarity function:

$$\text{Spatial Similarity}: \psi_s(s,m) = \frac{|s.R \cap m.R|}{|s.R|},$$

$$\text{Textual Similarity}: \psi_t(s,m) = \frac{\sum_{t \in s.T \cap m.T} \omega(t)}{\omega(s) = \sum_{t \in s.T} \omega(t)},$$

where $|s_i.R \cap m.R|$ is the size of the intersection of $s.R$ and $m.R$, $\omega(t)$ is the weight of term t which can be set as the inverse document frequency (idf) of t and $\omega(s)$ is the total token weight of subscription s. Existing spatial keyword search methods also utilize parameter α to leverage the textual relevancy and spatial proximity [8], and we can use these methods to tune parameter α.

Given a set of subscriptions $\mathcal{S} = \{s_1, s_2, \ldots, s_{|\mathcal{S}|}\}$, a message m and a threshold θ, a ranking based location-aware publish/subscribe system delivers m to $s \in \mathcal{S}$ if $\psi(s,m) \geq \theta$. We extend R^t-tree, R^{t+}-tree, and R^{t++}-tree to support the ranking semantics.

6.1 Extending R^t-tree

In the R^t-tree, besides the token sets in each node n, we also maintain the minimal token weight (denoted by $n.\min_\omega$) and the minimal region size (denoted by $n.\min_s$) of subscriptions under node n. In addition, we also keep a global token weight table to keep the weight of each token. Given a message m, for any subscription s under node n, we have

$$\psi_t(s,m) \leq \psi_t(n,m) = \frac{\sum_{t\, \in\, n.T\, \cap\, m.T} \omega(t)}{n.\min_\omega},$$

and

$$\psi_s(s,m) \leq \psi_s(n,m) = \frac{|n.R \cap m.R|}{n.\min_s}.$$

Thus we have

$$\psi(s,m) \leq \psi(n,m) = \alpha\psi_s(n,m) + (1-\alpha)\psi_t(n,m). \quad (4)$$

Obviously, for any node n, we can easily compute $\psi(n,m)$. If $\psi(n,m) \leq \theta$, we prune the node; otherwise we visit the children of the node. Thus we can traverse the R^t-tree and utilize this property to prune unnecessary nodes and iteratively we can get all answers.

6.2 Extending R^{t+}-tree

The R^t-tree will index larger numbers of tokens in each node and we extend R^{t+}-tree to prune unnecessary tokens. For each subscription s, its maximum spatial similarity to any message is 1. If s is similar to a message m, we can compute a lower bound of the textual similarity between s and m, i.e.,

$$\mathcal{B}_t^l = \frac{\theta - \alpha}{1-\alpha}. \quad (5)$$

Obviously if their textual similarity is smaller than the lower bound $\mathcal{B}_t^l$, we can prune the subscription. Based on this observation, we can prune some unnecessary tokens from s and only keep a *prefix token set* of s, denoted by $\textsc{Pre}(s) = \{s.T[1], s.T[2], \ldots, s.T[i]\}$, where i is the maximum number such that

$$\sum_{j>i} \omega(s.T[i]) \leq \mathcal{B}_t^l \cdot \omega(s).$$

Thus if message m does not contain a token in $\textsc{Pre}(s)$, the textual similarity between m and s will be smaller than $\mathcal{B}_t^l$ and thus m will not be similar to s and we can prune subscription s.

If a message has no spatial overlap with a subscription, we can deduce another tighter bound

$$\mathcal{B}_t^u = \frac{\theta}{1-\alpha} \quad (6)$$

Similarly, we can get a shorter prefix token set $\textsc{Pre}'(s) = \{s.T[1], s.T[2], \ldots, s.T[t]\}$, where t is the maximum number such that

$$\sum_{j>t} \omega(s.T[i]) \leq \mathcal{B}_t^u \cdot \omega(s).$$

Obviously, $\textsc{Pre}'(s)$ is a subset of $\textsc{Pre}(s)$. In this way, for each subscription s, we only index the tokens in its prefix sets $\textsc{Pre}(s)$ and $\textsc{Pre}'(s)$. As $\textsc{Pre}'(s) \subseteq \textsc{Pre}(s)$, we only need to maintain $\textsc{Pre}(s)$ and distinguish which tokens are in $\textsc{Pre}'(s)$ or $\textsc{Pre}(s)$. Similarly, for each node n, we also keep a prefix set $\textsc{Pre}(n)$ which is the union of prefixes of subscriptions under the node. Given a node n, (1) if message m has spatial overlap with n, we check whether m contains tokens in $\textsc{Pre}(n)$.[3] If no, we prune the node; otherwise we access the children of node n; (2) If the message has no spatial overlap with the node, we check whether m contains tokens in $\textsc{Pre}'(n)$. If no, we prune the node; otherwise we access the children of node n. We can utilize this property to find answers on R^{t+}-tree.

6.3 Extending R^{t++}-tree

We select the same representative tokens as the R^{t++}-tree. For each subscription s, we assign the ith token in s ($s.T[i]$) to its ancestor at the ith level. Different from the conjunctive semantics, we also maintain an upper bound $\mathcal{B}(s,n)$.

$$\mathcal{B}(s,n) = \frac{\sum_{j>i} \omega(s.T[j])}{\omega(s)}, \quad (7)$$

Next we discuss how to support ranking semantics. We first consider the case that $\theta > \alpha$. Given a message m and a tree node n, for each token in $t \in n.T \cap m.T$, we access the inverted list of token t. For each subscription s on the inverted list, if it is not added into the candidate set $\mathcal{C}$, we compute $\psi_s(s,m)$ in $\mathcal{O}(1)$ time. If $\alpha\psi_s(s,m) + (1-\alpha)\frac{\omega(t) + \mathcal{B}(s,n)}{\omega(s)} \geq \theta$, it is a candidate and we add $\langle s, \omega(t)\rangle$ into the candidate set. If s is already in the candidate set, we update its textual score to $\mathcal{C}(s) = \mathcal{C}(s) + \omega(t)$, and if $\alpha\psi_s(s,m) + (1-\alpha)\frac{\mathcal{C}(s) + \mathcal{B}(s,n)}{\omega(s)} \geq \theta$, it is a candidate and we add $\langle s, \mathcal{C}(s)\rangle$ into the candidate set; otherwise we remove it from the

3. If $\textsc{Pre}(n)$ (or $\textsc{Pre}(s)$) is empty, we need to visit the node (or subscription).

candidate set. After accessing the relevant leaf nodes, we get all the answers.

If $\theta \leq \alpha$, we need to made a minor change as a subscription may be similar to a message even if they do not share any common token. In this case, when accessing the root, we need to visit all the subscriptions in the inverted lists of the root. For other nodes, we do not need to make any changes.

Discussion. The operations to updates, selecting representative tokens and selecting token order are the same as those on R^t-tree, R^{t+}-tree, and R^{t++}-tree.

7 Experimental Study

We compared with state-of-the-art method IRTree [8]. We extended IRTree to support our problem as discussed in Section 2.3. We used two datasets. The first one was a real dataset Twitter. We collected 60 million tweets from May 2011 to August 2011, in which 13 million tweets had locations. We selected 10 million tweets with region information as subscriptions and used the others as messages. Each subscription contained an MBR and had 1-5 tokens selected from the tweets. The average token number of subscriptions was 3. The token distribution follows a Zipf's law. The number of subscription pairs with at least one common token is $2 * 10^{11}$ and the number of pairs with spatial overlap is $3.2 * 10^{11}$.

We generated four groups of messages as follows. (1) *Short Point Messages*: Each message contained 6-20 tokens and had a point location. (2) *Long Point Messages*: Each message contained 100-1000 tokens and had a point location. (3) *Short Range Messages*: Each message contained 6-20 tokens and had an MBR region. (4) *Long Range Messages*: Each message contained 100-1000 tokens and had an MBR region.

Each group contained 10,000 messages. We generated long messages by concatenating multiple tweets. We computed the average filtering time. The number of pairs of messages and subscriptions with at least a common token is $2.1 * 10^{10}$ and the number of pairs with spatial overlap is $2.3 * 10^{10}$. The average number of matched subscriptions of each message is 313. For long region messages, short region messages, long point messages, short point messages, the numbers are respectively 912, 431, 243, and 142.

We also used a synthetic dataset by combing Point of Interests (POIs) in USA and publications in DBLP. The USA dataset contained 17 million POIs and DBLP had 1.5 million publications. We generated MBRs from the POIs by selecting a POI as the center and extending a random width and height. Each subscription was generated by selecting an MBR and 1-5 tokens from DBLP. Each message was generated by selecting an MBR and a publication. We also generated four groups of messages and each group had 10,000 messages. Table 1 summarized datasets and index sizes, where the subscription length denotes the number of tokens in a subscription.

TABLE 1
Dataset Statistics

	Twitter	USA
Subscription number	10 million	10 million
Subscription length	1-5	1-5
Avg Subscription length	3	3
Subscription size	0.54 GB	0.65 GB
Token distribution	Zipf	Uniform
R^t-tree size	1.63 GB	1.76 GB
R^{t+}-tree size	0.79 GB	0.85 GB
R^{t++}-tree size	0.89 GB	0.92 GB

In the experiments we set $b = 25$ and $B = 50$. We get similar results on the two datasets. Due to space constraints, the results on the USA dataset are shown in Section 7.3 for comparing with other methods.

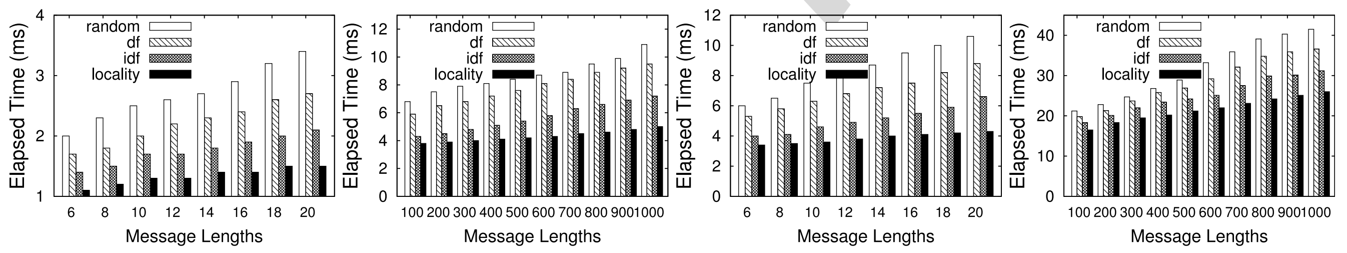
All algorithms were implemented in C++. All experiments were run on a Windows 2008 machine with an Intel Core E5410 2.33 GHz CPU and 16 GB RAM.

7.1 Evaluating Different Sorting Strategies

We evaluated different sorting strategies: random, df, idf, and locality-aware (We generated 10,000 grids and used the idf order) on the R^{t+}-tree as discussed in Section 4.2. The R^{t+}-tree sizes for the four methods were respectively 0.76, 0.72, 0.79, and 0.83 GB. This is because the df order shared many tokens in upper-level nodes and the idf order and the locality-aware method shared few tokens. Fig. 7 shows the results. We can see idf outperformed df which in turns was better than random as idf can prune many unnecessary nodes as infrequent tokens were on the upper-level nodes which had low probability to be contained in messages. df reduced token-set sizes by sharing common tokens, thus df outperformed random. The locality-aware method was better than df and idf as it considered the locality-aware token distributions to do pruning. For example, in Fig. 7c, for messages with 20 tokens, random and idf took more than 9 milliseconds, idf took 6 milliseconds, and the locality-aware method took 4 milliseconds.

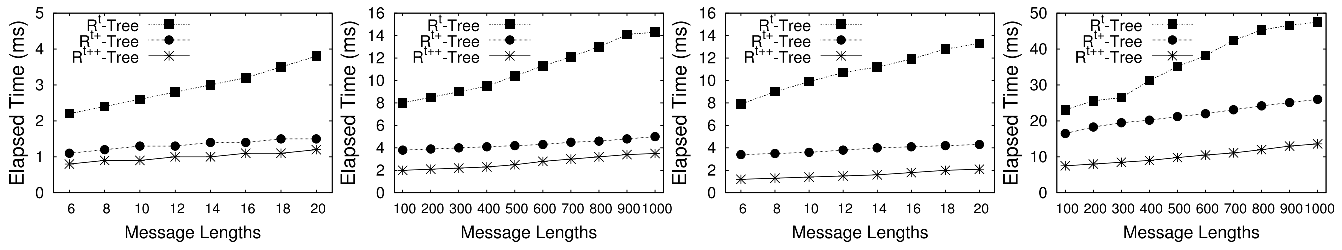
7.2 Evaluating R^t-tree with Different Token Sets

We evaluated R^t-tree with different token sets, i.e., R^t-tree with token sets, R^{t+}-tree with representative tokens, R^{t++}-tree with multiple representative tokens. For R^{t+}-tree and R^{t++}-tree, we used the locality-aware method. Fig. 8 shows



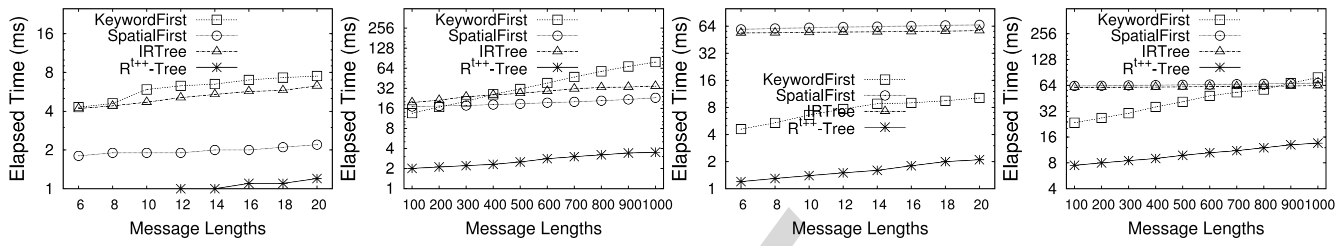
(a) Short Point Messages (b) Long Point Messages (c) Short Range Messages (d) Long Range Messages

Fig. 7. Evaluation on different sorting strategies using R^{t+}-tree on the Twitter dataset.



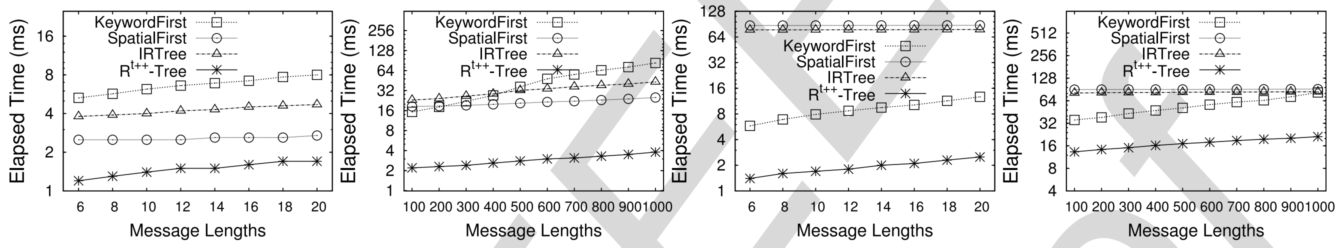
(a) Short Point Messages (b) Long Point Messages (c) Short Range Messages (d) Long Range Messages

Fig. 8. Evaluation on R^t-tree by using different token sets on the Twitter dataset.



(a) Short Point Messages (b) Long Point Messages (c) Short Range Messages (d) Long Range Messages

Fig. 9. Comparison with existing studies on the Twitter dataset.



(a) Short Point Messages (b) Long Point Messages (c) Short Range Messages (d) Long Range Messages

Fig. 10. Comparison with existing studies on the USA dataset.

the results. We can see that R^{t++}-tree outperformed R^{t+}-tree which was better than R^t-tree. This is because R^{t++}-tree had larger pruning power and did not involve an expensive verification step. R^{t+}-tree reduced the token-set sizes and decreased the number of candidates against R^t-tree, thus it achieved higher performance than R^t-tree. For example, in Fig. 8d, for messages with 1,000 tokens, R^t-tree took 50 milliseconds, and R^{t+}-tree decreased the time to 28 milliseconds, and R^{t++}-tree further reduced the time to 12 milliseconds. For short messages, e.g., tweets, in Figs. 8a and 8c, R^{t++}-tree only took 1-2 milliseconds.

7.3 Comparison with Existing Methods

We compared our best method R^{t++}-tree with existing approaches, the keyword-first method [31], the spatial-first method [27], and state-of-the-art spatial keyword search method IRTree [8] as discussed in Section 2.3. All algorithms employed an in-memory setting. Figs. 9 and 10 show the results on the Twitter and USA datasets respectively.

An observation is that the spatial-first method outperformed the keyword-first method for point messages (Figs. 9a, 9b, 10a, 10b). The reason is that the spatial-first method efficiently found candidate nodes using spatial indexes while the keyword-first method had no spatial pruning power. Another observation is that the spatial-first method had lower performance than the keyword-first method for range messages (Figs. 9c, 9d, 10c, 10d), as the spatial-first method had no textual pruning power.

In addition, notice that IRTree also achieved low performance and was even worse than the spatial-first method. There are two main reasons. First, it associated each R-tree node with a rather large inverted index and it was very expensive to traverse the R-tree by using the large inverted index. Second, it was designed for spatial keyword search and had to access larger numbers of unnecessary nodes. Thus IRTree was inefficient for the filtering problem.

Our R^{t++}-tree based algorithm always achieved the highest performance for any types of messages, because R^{t++}-tree seamlessly integrated the spatial and textual information and had large pruning power.

7.4 Scalability

We evaluated the scalability of the R^{t++}-tree based algorithm by varying the numbers of subscriptions. Fig. 11 shows the results. We can see that our method scaled very well, and with the increase of the numbers of subscriptions, the elapsed time increased sublinearly. This is because even if the number of subscriptions increased, our indexes still pruned large numbers of unnecessary subscriptions.

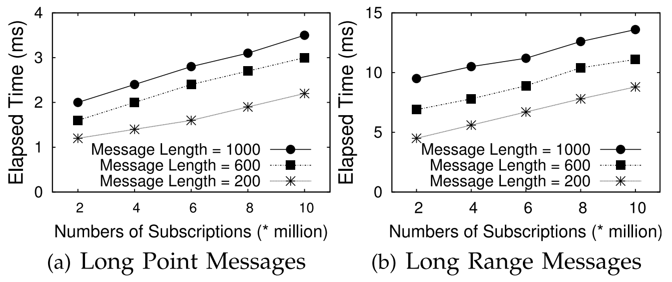


Fig. 11. Scalability of R^{t++}-tree on the Twitter dataset.

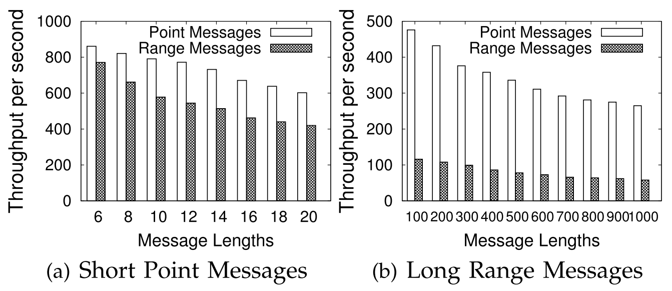


Fig. 12. Throughput of R^{t++}-tree on the Twitter dataset.

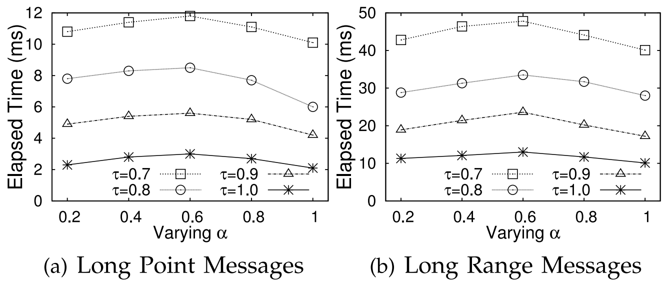


Fig. 13. Varying α on the Twitter dataset.

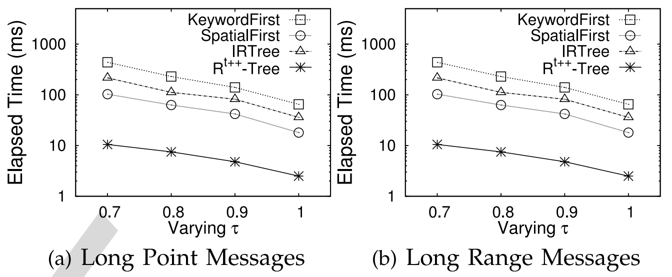


Fig. 14. Varying θ on the Twitter dataset.

7.5 Update and Query Throughput

We tested the throughput of the R^{t++}-tree based algorithm. We generated 100,000 queries with 10% insertions of subscriptions, 10 percent deletions of subscriptions and 80 percent new messages. We evaluated the throughput, i.e., the number of processed queries per second. Fig. 12 shows the result. We can see for short point messages, the throughput was 600-840; for long range messages and 60-120. We also evaluated the effect on updates of token frequencies. With the increase of frequencies, the performance slightly decreased. With the decrease of frequencies, the performance slightly increased. The new tokens had no effect on performance as they only enlarge hash table and efficiency of getting token frequency is not affected.

7.6 Ranking Semantics

We evaluated the ranking semantics. We first varied the parameter α and evaluated the average filtering time. Fig. 13 shows the result. We can see that with the increase of α, the performance first increased and then decreased, because for smaller α, the textual relevancy is more important, we can use prefix filtering to do effective pruning; for larger α, the spatial relevancy is more important, we can use the spatial filtering technique to prune irrelevant subscriptions.

Then we varied threshold θ and compared with baseline algorithms and IRTree. Fig. 14 shows the results. We can see that with the increase of θ, the performance decreased, because for larger θ, there are less results with similarity not smaller than the threshold and the filtering problem becomes easier.

8 Conclusion

In this paper, we study the location-aware publish/subscribe problem. We propose an effective index structure R^t-tree by integrating textual description into R-tree nodes. We develop a filter-and-verification framework and devise efficient filtering algorithms. We propose reducing the number of tokens in each node which not only reduces index sizes but improves performance. We devise an efficient algorithm to directly find answers without the verification step. We extend our algorithms to support both conjunctive queries and ranking queries. We discuss how to support ranking semantics. Experimental results on real datasets show our method achieves high performance and good scalability.

Acknowledgments

This work was partly supported by the National Natural Science Foundation of China under Grant No. 61272090 and 61373024, National Grand Fundamental Research 973 Program of China under Grant No. 2015CB358700 and 2011CB302206, Beijing Higher Education Young Elite Teacher Project under grant No. YETP0105, Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, the "NExT Research Center" funded by MDA, Singapore, under Grant No. WBS:R-252-300-001-490, and the FDCT/106/2012/A3.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra, "Matching events in a content-based subscription system," in *Proc. 18th Annu. ACM Symp. Principles Distrib. Comput.*, 1999, pp. 53–61.
[2] M. Altinel and M. J. Franklin, "Efficient filtering of XML documents for selective dissemination of information," in *Proc. 26th Int. Conf. Very Large Data Bases*, 2000, pp. 53–64.
[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proc. 21st ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2002, pp. 1–16.
[4] X. Cao, G. Cong, and C. S. Jensen, "Retrieving top-k prestige-based relevant spatial web objects," *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 373–384, 2010.
[5] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 373–384.
[6] X. Chen, Y. Chen, and F. Rao, "An efficient spatial publish/subscribe system for intelligent location-based services," in *Proc. 2nd Int. Workshop Distrib. Event-Based Syst.*, 2003, pp. 1–6.

[7] Y.-Y. Chen, T. Suel, and A. Markowetz, "Efficient query processing in geographic web search engines," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2006, pp. 277–288.

[8] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proc. VLDB*, vol. 2, no. 1, pp. 337–348, 2009.

[9] P. Costa and G. P. Picco, "Semi-probabilistic content-based publish-subscribe," in *Proc. 25th IEEE Int. Conf. Distrib. Comput. Syst.*, 2005, pp. 575–585.

[10] G. Cugola and J. E. M. de Cote, "On introducing location awareness in publish-subscribe middleware," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. Workshops*, 2005, pp. 377–382.

[11] Y. Diao and M. J. Franklin, "Query processing for high-volume XML message brokering," in *Proc. 29th Int. Conf. Very Large Data Base*, 2003, pp. 261–272.

[12] P. T. Eugster, B. Garbinato, and A. Holzer, "Location-based publish/subscribe," in *Proc. 4th IEEE Int. Symp. Netw. Comput. Appl.*, 2005, pp. 279–282.

[13] P. T. Eugster, B. Garbinato, and A. Holzer, "Pervaho: A specialized middleware for mobile context-aware applications," *Electron. Commerce Res.*, vol. 9, no. 4, pp. 245–268, 2009.

[14] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha, "Filtering algorithms and implementation for very fast publish/subscribe," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2001, pp. 115–126.

[15] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu, "Seal: Spatio-textual similarity search," *Proc. VLDB Endowment*, vol. 5, no. 9, pp. 824–835, 2012.

[16] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proc. Int. Conf. Data Eng.*, 2008, pp. 656–665.

[17] L. Fiege, F. C. Gärtner, O. Kasten, and A. Zeidler, "Supporting mobility in content-based publish/subscribe middleware," in *Proc. USENIX Int. Conf. Middleware*, 2003, pp. 103–122.

[18] P. W. Foltz and S. T. Dumais, "Personalized information delivery: An analysis of information filtering methods," *Commun. ACM*, vol. 35, no. 12, pp. 51–60, 1992.

[19] B. Garbinato, A. Holzer, and F. Vessaz, "Context-aware broadcasting approaches in mobile ad hoc networks," *Comput. Netw.*, vol. 54, no. 7, pp. 1210–1228, 2010.

[20] L. Golab and M. T. Özsu, "Issues in data stream management," *SIGMOD Rec.*, vol. 32, no. 2, pp. 5–14, 2003.

[21] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems," in *Proc. 19th Int. Conf. Sci. Stat. Database Manage.*, 2007, p. 16.

[22] K. W.-T. Leung, D. L. Lee, and W.-C. Lee, "Personalized web search with location preferences," in *Proc. Int. Conf. Data Eng.*, 2010, pp. 701–712.

[23] G. Li, J. Feng, and J. Xu, "Desks: Direction-aware spatial keyword search," in *Proc. Int. Conf. Data Eng.*, 2012, pp. 474–485.

[24] G. Li, Y. Wang, T. Wang, and J. Feng, "Location-aware publish/subscribe," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 802–810.

[25] J. Lu, Y. Lu, and G. Cong, "Reverse spatial and textual k nearest neighbor search," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 349–360.

[26] S. B. Roy and K. Chakrabarti, "Location-aware type ahead search on spatial databases: Semantics and efficiency," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 361–372.

[27] H. Samet, *Foundations of Multidimensional and Metric Data Structure*. San Mateo, CA, USA: Morgan Kaufmann, 2006.

[28] M. Sharifzadeh and C. Shahabi, "Approximate voronoi cell computation on spatial data streams," *VLDB J.*, vol. 18, no. 1, pp. 57–75, 2009.

[29] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, "Efficient continuously moving top-k spatial keyword query processing," in *Proc. Int. Conf. Data Eng.*, 2011, pp. 541–552.

[30] T. W. Yan and H. Garcia-Molina, "Index structures for information filtering under the vector space model," in *Proc. Int. Conf. Data Eng.*, 1994, pp. 337–347.

[31] T. W. Yan and H. Garcia-Molina, "Index structures for selective dissemination of information under the boolean model," *ACM Trans. Database Syst.*, vol. 19, no. 2, pp. 332–364, 1994.

[32] B. Yao, F. Li, M. Hadjieleftheriou, and K. Hou, "Approximate string search in spatial databases," in *Proc. Int. Conf. Data Eng.*, 2010, pp. 545–556.

[33] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *Proc. Int. Conf. Data Eng.*, 2009, pp. 688–699.

[34] D. Zhang, B. C. Ooi, and A. K. H. Tung, "Locating mapped resources in web 2.0," in *Proc. Int. Conf. Data Eng.*, 2010, pp. 521–532.

[35] R. Zhong, J. Fan, G. Li, K.-L. Tan, and L. Zhou, "Location-aware instant search," in *Proc. 21st ACM Int. Conf. Inf. knowl. Manage.*, 2012, pp. 385–394.

[36] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma, "Hybrid index structures for location-based web search," in *Proc. 21st ACM Int. Conf. Inf. knowl. Manage.*, 2005.

Minghe Yu received the bachelor's degree from the Department of Computer Science, Northeast University, China. She is currently working toward the PhD degree in the Department of Computer Science, Tsinghua University, Beijing, China. Her research interests include data integration and spatio-textual data query.

Guoliang Li received the PhD degree in computer science from Tsinghua University, Beijing, China in 2009. He is currently working as an associate professor in the Department of Computer Science, Tsinghua University, Beijing, China. His research interests mainly include data cleaning and integration, spatial databases, and crowdsourcing. He is a member of the IEEE.

Ting Wang received the bachelor's and the master's degree from the Department of Computer Science, Tsinghua University, Beijing, China. He is currently working at Google. His research interests mainly include data integration and spatio-textual data query.

Jianhua Feng received the BS, MS, and the PhD degrees in computer science from Tsinghua University. He is currently working as a professor of Department Computer Science in Tsinghua University. His main research interests include large-scale data management and analysis.

Zhiguo Gong received the PhD degree from the Department of Computer Science, Institute of Mathematics, Chinese Academy of Science, China. He is an associate professor in the Faculty of Science and Technology, University of Macau. His research fields include database systems, Web information retrieval, and Web mining.