A Cost-based Method for Location-Aware Publish/Subscribe Services

Minghe Yu Guoliang Li Jianhua Feng
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China.
yumh12@mails.tsinghua.edu.cn, liguoliang@tsinghua.edu.cn, fengjh@tsinghua.edu.cn

ABSTRACT

Location-based services have attracted significant attentions from both industry and academia, thanks to modern smartphones and mobile Internet. To provide users with gratifications, location-aware publish/subscribe has been recently proposed, which delivers spatio-textual messages of publishers to subscribers whose registered spatio-textual subscriptions are relevant to the messages. Since there could be large numbers of subscriptions, it is necessary to devise an efficient location-aware publish/subscribe system to enable instant message filtering. To this end, in this paper we propose two novel indexing structures, MBRTrie and PT-Quadtree. Using the indexes, we devise two filtering algorithms to support fast message filtering. We analyze the complexities of the two filtering algorithms and develop a cost-based model to judiciously select the best filtering algorithm for different scenarios. The experimental results show that our method achieves high performance and significantly outperforms the baseline approaches.

1. INTRODUCTION

Location-based services (LBS) such as Foursquare[1] and Yelp[2] have been extensively deployed in many systems and widely accepted by the Internet users. LBS includes two important models: user-initiated model and server-initiated model. The former, aka location-aware keyword search, returns the relevant answers for each user-initiated spatio-textual query (which includes the query location and query keywords) [7, 5]. For example, considering a user who wants to find a hotel nearby, she issues a spatio-textual query with her current location and keywords "`hotel two-beds room`" to an LBS system, which returns relevant answers by considering the spatial proximity and textual relevancy. The latter, aka location-aware publish/subscribe, delivers the spatio-textual messages of publishers to relevant subscribers.

In this paper, we focus on location-aware publish/subscribe. The subscribers register spatio-textual subscriptions. When a publisher posts a spatio-textual message, the system instantly delivers the message to the subscribers whose registered subscriptions are relevant to the message. To evaluate the relevancy between spatio-textual messages and subscriptions, we need to combine spatial proximity and textual relevancy (see Section 2 for more details).

There are many real-world applications that require the location-aware publish/subscribe services. For example, consider a Groupon system. The subscribers (e.g., Groupon customers) register their spatio-textual subscriptions (e.g., "`The Avengers`" at `Melbourne University`) to the system. For each message posted by the publisher (e.g., a Groupon message about movie "`The Avengers: Age of Ultron`", at Melbourne Downtown), the system delivers the message to relevant subscribers. Location-aware publish/subscribe has many other real applications, e.g., location-aware news delivery, location-aware advertisements, and location-aware market analysis [14, 3].

A publish/subscribe system requires to support large numbers of subscriptions and a big challenge is to enable instant message filtering [14, 3, 24]. Although Li et al. [14] proposes an R^t-tree based method, which adds some selected tokens from spatio-textual subscriptions into R-tree nodes and uses the R^t-tree to filter messages, it cannot meet the high-performance requirement, because it has low pruning power on the textual part, especially for upper-level R^t-tree nodes that contain too many selected tokens.

To address this challenge, we propose two novel spatio-textual index structures, MBRTrie and PT-Quadtree. MBRTrie constructs a trie structure based on the textual description and integrates spatial information into trie nodes while PT-Quadtree utilizes a quadtree to index spatial information and adds the textual descriptions into the quadtree nodes. Using the indexes, we devise two efficient filtering algorithms. MBRTrie and PT-Quadtree have their own superiorities: MBRTrie is efficient for the messages with a small number of tokens while PT-Quadtree is efficient for messages with small spatial regions. To achieve high performance for various settings, we analyze the complexities of the two algorithms and develop a cost-based model to judiciously select the best filtering algorithm for various scenarios.

To summarize, we make the following contributions.

(1) We propose two spatio-textual index structures, MBRTrie and PT-Quadtree. Based on the two index structures, we propose two efficient filtering algorithm.

(2) We analyze the complexities of the two filtering algo-

[1]http://foursquare.com
[2]http://www.yelp.com/

CIKM'15, October 19–23, 2015, Melbourne, Australia.

DOI: http://dx.doi.org/10.1145/2806416.2806427.

rithms and develop a cost-based model that selects the best algorithm to filter a message.

(3) Based on the cost model, we develop a new filtering algorithm and develop several pruning techniques.

(4) We have implemented our method. The experimental results on real datasets show our method achieves high performance and significantly outperforms existing approaches.

The rest of this paper is organized as follows. In Section 2, we formulate the problem and review related works. We present an MBRTrie index and develop a filter-verification framework in Section 3 while PT-Quadtree index structure is proposed in Section 4. Then we devise a cost-based algorithm in Section 5. The experimental results are shown in Section 6. And finally we conclude in Section 7.

2. PRELIMINARIES

We first formalize the location-aware publish/subscribe problem and then review related works.

2.1 Problem Formulation

In a location-aware publish/subscribe system, publishers post spatio-textual messages and subscribers register subscriptions to capture their interests. A subscription s includes a textual description $s.T$ and spatial information $s.R$, denoted by $s = \langle T, R\rangle$. The spatial information is used to capture a subscriber's most interested region. In this paper we use the well-known minimum bounding rectangle (MBR) to denote a region $s.R$. We use the bottom-left corner and top-right corner to describe an MBR, denoted by $s.R = \langle (x_{bl}, y_{bl}), (x_{tr}, y_{tr})\rangle$. Especially if $x_{bl} = x_{tr}$ and $y_{bl} = y_{tr}$, the region is reduced to a point. The textual description is used to capture a subscriber's content interests, denoted by a set of tokens $s.T = \langle t_1, t_2, \cdots, t_{|s.T|}\rangle$.

A message m is denoted by $m = \langle T, R\rangle$ in which $m.T$ and $m.R$ have the same meaning as those of subscriptions. If the spatial information of a message is a point, we call it *point message*; otherwise it is called a *range message*.

Given a subscription s and a message m, s could be an answer of m if s and m have spatial overlap and all the tokens in s are contained in m[3], i.e. they satisfy

(1) *Spatial constraint*: $m.R \cap s.R \neq \phi$ and;

(2) *Textual constraint*: $m.T \supseteq s.T$.

Based on the notations, we formalize the location-aware publish/subscribe problem.

DEFINITION 1 (LOCATION-AWARE PUBLISH/SUBSCRIBE). *Given a set of subscriptions $\mathcal{S} = \{s_1, s_2, \cdots, s_{|\mathcal{S}|}\}$ and a message m, a location-aware publish/subscribe system delivers m to $s_i \in \mathcal{S}$ if $m.R \cap s_i.R \neq \phi$ and $m.T \supseteq s_i.T$.*

For example, consider the twelve subscriptions in Figure 1. For a point message $m_p = (\{b, c, d, e, f\}, (26, 14))$, subscription $s_4 = (\{b, c, d\}, \langle(20, 10), (28, 18)\rangle)$ is an answer of m_p. $s_3 = (\{b, c, d\}, \langle(20, 32), (35, 35)\rangle)$ is not an answer as it does not contain the point of m_p. $s_1 = (\{a, b, c\}, \langle(25, 0), (30, 20)\rangle)$ is also not an answer as it has a token "a" which does not appear in m_p. We deliver subscriptions s_4, s_7, s_{10} to message m_p. For a range message $m_r = (\{a, c, d, e\}, \langle(10, 10), (40, 40)\rangle)$, we deliver the subscriptions s_2, s_{10}, s_{11} to m_r.

[3]Our method can efficiently support other metrics, and due to space constraints, we omit the details.

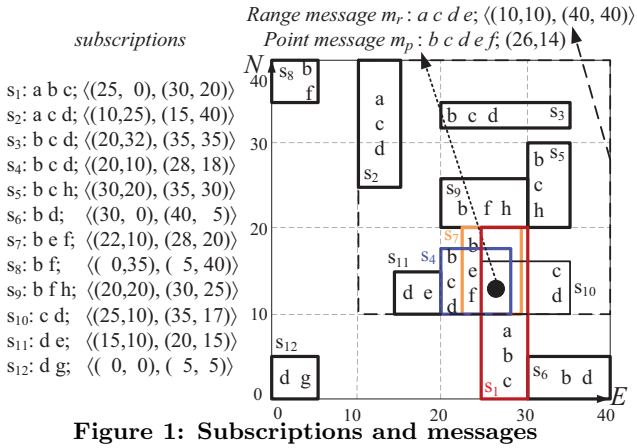


Figure 1: Subscriptions and messages

2.2 Related Work

Location-Aware Publish/Subscribe. Li et al. [14] proposed the R^t-tree to support location-aware publish/subscribe, which extended the R-Tree by selecting some representative tokens from subscriptions and adding them into R-tree nodes to enable textual pruning. However, for upper-level nodes, there are many tokens and thus it has low pruning power on textual part. We have compared with this approach and our method significantly outperformed it (see Section 6).

Hu et al. [11] studied the parameterized location-aware publish/subscribe, which requires subscribers to specify parameters to enable personalize filtering. Chen et al. [3] studied the problem of matching boolean range continuous queries to streaming objects. Guo et al. [9] studied on filtering dynamic streams for continuous moving boolean subscriptions. Obviously they are different from our problem. Wang et al. [19] proposed a cost-based method to support spatial keyword queries using a tree structure. Different from our works, they utilized keyword partition and space partition in one tree structure when constructing index for queries based on expected matching cost. And they computed the cost based on the number of queries associated to each partition and the probability that whether the partition is explored during object matching, instead of the complexity of filter and verification steps.

Location-Aware Keyword Search: There are several studies on location-aware keyword search [7, 25, 13, 2, 16, 12]. Felipe et al. [7] proposed IR^2-Tree which combines an R-tree with text signatures to find top-k answers. Cong et al. [2] presented a location-aware textual search model based on integrating R-tree and inverted files. Lu et al. [16] studied on supporting reverse spatial textual search. All these works designed algorithms to find answers based on the given location and keywords. For the region-based keyword search problem, Jin et al. [12] proposed a hybrid model to return objects that contain related keywords within a given region. Zhou et al. [25] and Hariharan et al. [10] proposed different strategies for combining R^*-tree and inverted lists.

In addition, location-aware keyword search has been applied into different scenarios. For example, Yao et al. [23] presented a model to answer similarity string queries in spatial databases. Chen et al. [4] studied on query processing in geographic search engines based on spatial keyword search. Roy and Chakrabarti [17] designed a location-aware type ahead search. Wu et al. [20] studied on processing continuously moving spatial keywords queries.

Different from location-aware keyword search, location-aware publish/subscribe services adopt a push model which delivers message to relevant subscribers.

Traditional Publish/Subscribe Services: Foltz and Dumais [8, 21, 22] studied on selecting textual information under space model from IR perspective. Fabret et al. [6] designed a "processor cache conscious" algorithm for matching subscriptions. Liu et al. [15] utilized synopses to construct an index to support the publish/subscribe problem.

3. MBRTRIE BASED METHOD

In this section, we first propose an index MBRTrie in Section 3.1, and devise a filtering framework in Section 3.2. Then we develop efficient filtering algorithms in Section 3.3.

3.1 MBRTrie Index Structure

Before index construction, we first fix a global order for tokens in subscriptions, e.g., the alphabetical order. Then for each subscription, we sort its tokens based on this order. Given a sorted token set $T = \langle t_1, t_2, \cdots, t_{|T|}\rangle$, $\langle t_1, t_2, \cdots, t_i\rangle$ is called a prefix of $\langle t_1, t_2, \cdots, t_j\rangle$ if $i \leq j \leq |T|$. A token set $\langle t_1, t_2, \cdots, t_x\rangle$ is smaller than token set $\langle t'_1, t'_2, \cdots, t'_y\rangle$ if $t_i < t'_i$ and $t_j = t'_j$ for $j < i$. We also sort the subscriptions based on their token-set order, and associate each subscription with a subscription ID.

We construct an MBRTrie for subscriptions as follows. For each subscription, we utilize its token set to build a unique path from the root to a leaf in which each trie node is associated with a token. The root and leaf have an empty token ϵ. For simplicity, a trie node is interchangeably used with its corresponding token in this paper. Based on this structure, the subscriptions with a common prefix of token sets share the same ancestors. And children of a node are sorted by the corresponding token. Each leaf has an inverted list of subscriptions' IDs which contains the corresponding token set. As subscriptions are sorted by the token-set order, we associate each node n with a subscription ID range $[n_l, n_u]$, where n_l and n_u respectively denote the minimum and maximal IDs of subscriptions under this node (which are subscriptions on the inverted lists of its leaf descendants).

For each node n, we also associate it with a *node MBR*, which is the minimum bounding rectangle of MBRs of subscriptions under node n, denoted by n_{mbr}. And if the MBR of a message m has no overlap with n_{mbr}, the subtrie rooted at node n can be pruned since all the subscriptions under it have no spatial overlap with m.

For example, Figure 2 gives an MBRTrie of subscriptions in Figure 1 with alphabetical order. The MBR of node 1 is $\langle(10, 0), (30, 40)\rangle$, which is the minimum bounding rectangle of MBRs of subscriptions under node 1, i.e., s_1 and s_2. Node 8 has a subscription range $[s_3, s_9]$. That means subscriptions s_3 to s_9 are under this node. In other words, subscriptions with token "b" (node 8) as the first token must be in $[s_3, s_9]$.

Space Complexity: Suppose the average token number of subscriptions is $\mathcal{S}_{avg}$, which is usually very small, e.g., 3. The number of nodes in the MBRTrie is at most $\mathcal{S}_{avg} \times |\mathcal{S}|$. As many token sets share prefix, the real number is smaller than it. Thus with the total length of inverted lists $|\mathcal{S}|$, the space complexity of MBRTrie is $\mathcal{O}(\mathcal{S}_{avg} \times |\mathcal{S}|)$.

Updates: For a new subscription, we only need to insert it into MBRTrie and update the MBRs and subscription ranges of the nodes on the path from the root to the new inserted leaf. As subscription IDs are sorted in order, we can reserve some IDs for accommodating future insertions. For deletions, we can use a special mark to denote the deletions.

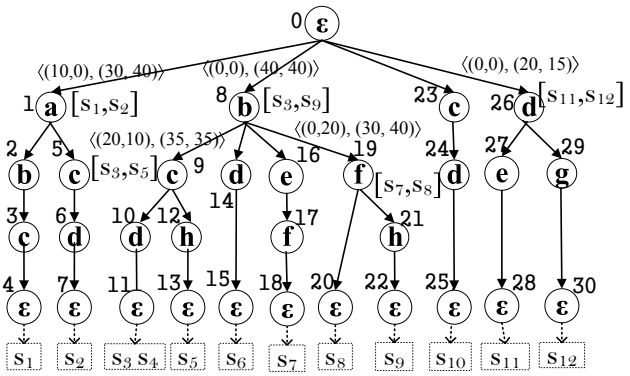


Figure 2: MBRTrie index by an alphabetical order

3.2 A Filtering Framework

To achieve high performance, we want to only visit a small number of relevant trie nodes. Thus in this subsection, we will discuss how to filter MBRTrie nodes.

When traversing MBRTrie, we only need to visit the nodes having leaf descendants which contain subscriptions as results of message. However an MBRTrie node may have a lot of leaf descendants and checking whether it satisfies this condition is expensive. Therefore we propose a filter-and-verification framework. The first step identifies a set of candidate nodes (there is an answer under the node) and the second step verifies the subscriptions on the inverted lists of leaf descendants of these candidate nodes.

Step 1 - Filter: For a trie node n, we prune node n, if (1) MBR Filter: $n_{mbr} \cap m.R = \phi$, which means any subscriptions under node n have no spatial overlap with m; or (2) Token Filter: $n \notin m.T$ (The token of n is not in $m.T$). As all subscriptions under node n containing this token which does not appear in m, it invalidates the textual constraint.

Step 2 - Verification: For each leaf node surviving in the filter step, the subscriptions on its inverted list are candidate answers of message m. All of them must already satisfy the textual constraint since all the tokens in their ancestors appear in the message (otherwise this node should be pruned). However the subscriptions may not satisfy the spatial constraint because the node MBR contains some regions that have no overlap with the subscriptions in its leaf descendants. Therefore we need a verification step to do spatial checking for the subscriptions.

In the verification step, we examine all subscriptions on the inverted lists of leaf candidate nodes. And time complexity is $\mathcal{O}(\sum_l |\mathcal{I}_l|)$, where l denotes a leaf candidate node and $\mathcal{I}_l$ is the inverted list of l. In the filter step, we traverse MBRTrie to find candidate nodes. Given a node, the MBR filter checks whether its node MBR has spatial overlap with the message with complexity $\mathcal{O}(1)$. For token filter, we find all the trie nodes which appear in the message. Next we introduce several algorithms to efficiently find such nodes.

3.3 Filtering Algorithms

Given a message m, we first sort the tokens in message m based on a global order (e.g., alphabetical order) and remove the duplicates. Let $T = \langle t_1, t_2, \cdots, t_N\rangle$ denote the sorted token set of m. Next we propose three algorithms to implement the token filter.

ProbeTokenSet - Use trie nodes to probe token sets: The first method traverses MBRTrie in pre-order. For each node n, we check whether it satisfies two filters. If not, we prune node n; otherwise we access its children. Iteratively, we can find all leaf candidate nodes.

As tokens in T are sorted, we can do a binary search to check whether a node n appears in T. The time complexity for checking is $\mathcal{O}(\log N)$, where N is the number of tokens in T. In the worst case, we need to enumerate all trie nodes, thus the worst-case complexity is $\mathcal{O}(M \times \log N)$, where M is the number of nodes in MBRTrie. Next we analyze the average-case complexity.

Suppose there are totally D distinct tokens which are independent. That is given a node, any token has the same probability to appear as a child of it, which is $\frac{1}{D}$. Suppose there are L_i nodes in the i-th level (the level of the root is 0). For the root, the probability to visit its children in the first level is 1. As there are L_1 nodes in the first level, the time complexity for the first-level nodes is $\mathcal{O}(L_1 \times \log N)$. Each node in this level has $\frac{N}{D}$ probability to appear in T, which means the probability to access nodes in the second level is $\frac{N}{D}$. As there are L_2 nodes in the second level, the average-case complexity for the second-level nodes is $\mathcal{O}(L_2 \times \frac{N}{D} \times \log N)$. Similarly the average-case time complexity for the i-th level nodes is $\mathcal{O}(L_i \times (\frac{N}{D})^{i-1} \times \log N)$. Thus the average-case time complexity for all nodes is

$$\mathcal{O}\big(\log N \times \sum_{i=1}^{H_t}(L_i \times (\tfrac{N}{D})^{i-1})\big),$$

where H_t is the height of MBRTrie.

In practice, $N \ll D$, and for $i \geq 3$, $L_i \times (\frac{N}{D})^{i-1} \times \log N$ is close to 0. The average-case time complexity can reduce to

$$\mathcal{O}\big((L_1 + \tfrac{L_2 \times N}{D}) \times \log N\big).$$

ProbeTrie - Use the tokens to probe trie nodes: The second method visits tokens in T in order. For each token $t_i \in T$, it checks whether the root has a child with token t_i and has spatial overlap with m. If there is no such node, it breaks and visits the next token t_{i+1}; otherwise, it locates the node with token t_i, and as tokens in T are sorted in order, next we repeat this step for each token after t_i in T. Iteratively, it can find all leaf candidate nodes.

To check whether a trie node has a child with token t_i, we can do a binary search among all the children of this node. Suppose the average fanout of a node is F. The complexity to do the checking is $\mathcal{O}(\log F)$. In the worst case for each combination of tokens in T, we need to do a binary search on the trie, thus the worst-case complexity is

$$\mathcal{O}\Big(\textstyle\sum_{i=1}^{N}\binom{N}{i} \times \log F = 2^N \times \log F\Big).$$

Next we give the average-case complexity. The fanout of the root is L_1. Thus the average-case time complexity for the first-level nodes is $\mathcal{O}(N \times \log L_1)$. Considering the i-th level nodes, their parents are in $(i-1)$-th level and have $(\frac{N}{D})^{i-1}$ probability in T. The average fan-out of $(i-1)$-th level nodes is $F_{i-1} = \frac{L_i}{L_{i-1}}$. The average-case time complexity for the i-th level nodes is $\mathcal{O}\Big(logF_{i-1} \times L_{i-1} \times (\frac{N}{D})^{i-1}\Big)$. So the average-case complexity is

$$\mathcal{O}\Big(N \times \textstyle\sum_{i=1}^{H_t}\big(\log F_{i-1} \times L_{i-1} \times (\frac{N}{D})^{i-1}\big)\Big).$$

In practice, for $i \geq 3$, $\log F_{i-1} \times L_{i-1} \times (\frac{N}{D})^{i-1}$ is close to 0. Thus the average-case time complexity can reduce to

$$\mathcal{O}\big(N \times \log L_1 + N \times \log F_1 \times L_1 \times \tfrac{N}{D}\big).$$

Algorithm 1: MBRTrie $(\mathcal{S}, m)$

Input: $\mathcal{S}$: The subscription set; m: A message
Output: $\mathcal{R}$: Answers of m

```
1 begin
2     Fix a global token order ;
3     Build an MBRTrie with root r ;
4     T ← sorted distinct token set of m ;
5     Initialize a candidate node set CN;
6     if r_mbr ∩ m.R ≠ φ then
7         MBRTrie-Filter (r, m.T m, CN) ;
8         R = MBRTrie-Verify (m, CN) ;
9 end
```

Function MBRTrie-Filter(n, T_n, m, $\mathcal{CN}$)

Input: n: A trie node; T_n: Token set; m: A message
$\mathcal{CN}$: Candidate leaf node set

```
1 begin
2     if n is a leaf node then CN ← n ;
3     P = SelectPair (n, T_n, m) ;
4     for each pair ⟨c, T_c⟩ ∈ P do
5         MBRTrie-Filter (c, T_c, m, CN);
6 end
```

Function SelectPair(n, T_n, m)

Input: n: A trie node; T_n: Token set; m: A message
Output: $\mathcal{P}$: Candidate node set

```
1 begin
2     if |T_n| × log|n| < |n| × log|T_n| then
3         for each token t of T_n do
4             Use t to do a binary search on n's children;
5             if child c with label t & c_mbr ∩ m.R = φ then
6                 P∪ = ⟨c, T_c⟩; //T_c : tokens after c in T_n
7     else
8         for each child c of n & c_mbr ∩ m.R = φ do
9             Use c to do a binary search on T_n;
10            if c ∈ T_n then
11                P∪ = ⟨c, T_c⟩; //T_c : tokens after c in T_n
12 end
```

Function MBRTrie-Verify(m, $\mathcal{CN}$)

Input: m: A message; $\mathcal{CN}$: Leaf candidate node set
Output: $\mathcal{R}$: Answers of m

```
1 begin
2     for each node n ∈ CN do
3         for each subscription s ∈ I_n do
4             if s.R ∩ m.R ≠ φ then R ← s ;
5 end
```

Figure 3: MBRTrie algorithm

A Cost-based Method: Notice that upper-level nodes usually have many children, thus it is efficient to use tokens to do a binary search on the trie structure. On the contrary, it is more efficient to use trie nodes to probe tokens in message on the lower-level trie nodes which have few children. Based on this observation, we propose a cost-based method.

Consider a candidate node n. There exists one and only one token in T that matches n (as tokens in T are distinct).

Suppose it is t_i. Let $T_n = \{t_{i+1} \cdots t_N\}$. We call $\langle n, T_n \rangle$ a candidate pair if n satisfies the spatial constraint. If $\langle n, T_n \rangle$ is a candidate pair, we try to find candidate pairs from n's children. As tokens in T are sorted, n's children can only match tokens in T_n. Based on this observation, we propose an extension-based method to find candidate pairs. Let $\mathcal{P}$ denotes the candidate-pair set for n's children, $|n|$ denotes the number of n's children. We compute $\mathcal{P}$ as follows.

(1) If $|T_n| \times \log |n| < |n| \times \log |T_n|$, we use each token $t_j \in T_n$ to probe n's children. If a child c matches t_j and $c_{mbr} \cap m.R \neq \phi$, $\langle c, T_c = \langle t_{j+1}, \cdots, t_N \rangle \rangle$ is a candidate pair and added into $\mathcal{P}$.

(2) Otherwise, we use each child of n to probe token set T_n. For a child c, if c matches a token $t_j \in T_n$ and $c_{mbr} \cap m.R \neq \phi$, $\langle c, T_c = \langle t_{j+1}, \cdots, t_N \rangle \rangle$ is a candidate pair and added to $\mathcal{P}$.

Next we repeat the above steps for pairs in $\mathcal{P}$. Iteratively we can find all candidate pairs. And based on the candidate pairs, we can find the leaf candidate nodes.

Figure 3 illustrates the MBRTrie algorithm. It first fixes a global token order and constructs an MBRTrie (line 2- 3). Then it calls function MBRTrie-Filter to find candidate nodes (line 7). Finally it verifies the candidates and gets the answers by calling function MBRTrie-Verify (line 8).

MBRTrie-Filter first checks whether the node in a candidate pair is a leaf node. If yes, we select it into leaf candidate node set (line 2). Next MBRTrie-Filter adaptively selects a better method to find candidate pairs by calling function SelectPair (line 3). If MBRTrie-Filter finds a candidate pair, it recursively finds candidate pairs based on the current pair (line 5).

Time Complexity: As we select a better method to find trie nodes, the average-case complexity of the filter step is

$$\mathsf{Cost}_T^{\mathcal{F}} = \mathcal{O}\Big(\sum_{i=1}^{H_t} \min\big(\log N \times L_i, N \times \log F_{i-1} \times L_{i-1}\big) \times \big(\frac{N}{D}\big)^{i-1}\Big). \quad (1)$$

Suppose there are cand_T candidate nodes and the inverted lists of leaf nodes are $\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_{\mathsf{cand}_T}$. In the verification step, the time complexity is

$$\mathsf{Cost}_T^{\mathcal{V}} = \mathcal{O}(\sum_{i=1}^{\mathsf{cand}_T} |\mathcal{I}_i|). \quad (2)$$

An algorithm should satisfy (1) completeness: any subscription satisfying the spatial constraint and the textual constraint must be found by the algorithm; and (2) correctness: any subscription found by the algorithm must satisfy the spatial constraint and the textual constraint. The MBRTrie satisfies the two properties as formalized in Theorem 1.

Theorem 1. *The MBRTrie algorithm satisfies completeness and correctness.*[4]

For example, consider message $m_p = (\{b, c, d, e, f\}, (26, 14))$ and the MBRTrie in Figure 2. As there are 5 tokens in the message and the root has 4 children, we use trie nodes to probe the token set and find three candidate nodes 1, 8, 23. Node 26 is pruned as its MBR does not contain the message point. For node 1, we find an answer s_1 in its child node 2. For node 8, we select its children 9, 16 and prune nodes 14 and 19. Iteratively we can find all answers for the message m_p. Here we prune sixteen nodes and only visit fifteen nodes in MBRTrie, and the visisted nodes are nodes 0∼4, 8∼11, 16∼18, and 23∼25.

[4]We omit proofs of Lemmas and Theorems due to space constraints.

4. PT-QUADTREE BASED METHOD

The node MBRs of the upper-level nodes in the MBRTrie cover large numbers of subscription MBRs and may involve many unnecessary regions. Thus they may have low spatial pruning power. To address this issue, we propose a quadtree-based index structure in Section 4.1, and then introduce efficient filtering algorithms in Section 4.2.

4.1 Pivotal-Token Quadtree

To facilitate spatial pruning, we construct a qudatree for MBRs of subscriptions. Firstly we generate a minimal region to cover all subscription MBRs and select the center of the region as root node, and then divide the region into four subregions. Recursively, we divide each subregion into four small regions. For each region, we terminate its division and take it as a leaf node if (1) the number of subscription MBRs that have overlap with the region is smaller than a threshold τ_{num}; or (2) the area of the region is smaller than a threshold τ_{area}. For each leaf node, we maintain an inverted list of sorted IDs of subscriptions whose MBRs have overlap with the region.

If many MBRs have overlaps, they may appear in many inverted lists of leaf nodes. To address this issue, we can use the MX-CIF quadtree implementation [18], which associates each MBR with the quadtree node corresponding to the smallest region that contains the MBR. In other words, the subscription IDs can be associated with both leaf and non-leaf nodes, and each subscription ID appears in one and only one inverted list. Moreover, given an extra space buffer, we can select some regions and push down the subscription IDs in these regions to their leaf descendant regions.

To support textual pruning in the quadtree, we integrate textual description into nodes. A straightforward method associates a node with all the distinct tokens of subscriptions that have overlap with it. Based on this method, given a message m, we traverse the quadtree from root. For each node, if it has overlap with $m.R$ and there exists a token associated with the node appearing in $m.T$, we access the node. Otherwise we prune the node, as it does not satisfy the spatial or textual constraint.

For example, consider the quadtree in Figure 4. Subscriptions s_3, s_5, s_9 are in node 1. The token set associated with node 1 should be the union of their token sets, i.e., $\{b, c, d, f, h\}$. For range message $m_r = (\{a, c, d, e\}; \langle (10, 10), (40, 40) \rangle)$ in Figure 1, as node 1 has spatial overlap with m_r and contains a token "c" that appears in the token set, we need to access the node.

In the straightforward method, the number of tokens associated with a node is very large and the pruning power is low. For instance, in the last example we should prune node 1, because although node 1 shares token "c" with message m_r, all subscriptions under this node contain another token "b" which does not appear in the message. This is not achieved when we access node 1. To address this issue, we propose a pivotal-token quadtree to reduce the number of tokens associated with nodes. Next we discuss how to select and associate pivotal tokens with each node.

For a leaf node, we generate its pivotal-token set by selecting *a single token* from each subscription on its inverted list to represent it. For an internal node, its pivotal-token set is the union of its children's pivotal-token sets. Obviously if the

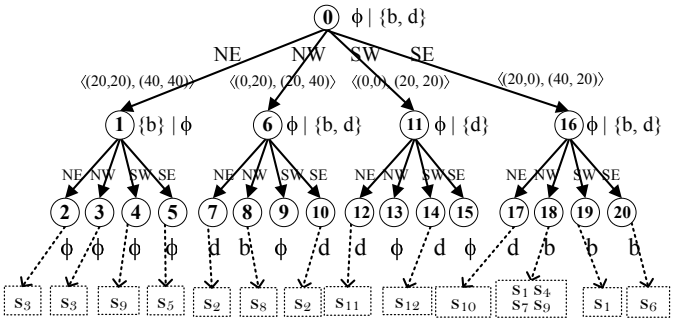


Figure 4: PT-Quadtree index structure

pivotal-token set has no common token with the message, we can prune the node, as stated in Lemma 1.

LEMMA 1. *Given a quadtree node n and a message m, if n's pivotal-token set has no common token with $m.T$, we can prune the node.*

To save the space and improve the performance, we want to minimize the pivotal-token set. However like the minimum hitting set problem, this is an NP-hard problem [1]. Thus we propose a greedy algorithm. First we select the largest df token into the pivotal-token set and remove the subscriptions contain it. Then the largest df token in the rest of subscriptions is put into pivotal-token set. Iteratively we can generate the pivotal-token set for leaf nodes until no subscription left. For each internal node, its pivotal-token set is generated by computing the union of its children's sets.

We can further remove tokens for internal nodes as follows. Given a node n, if all its children contain a token t, we will keep t in n, and remove it from all four children. In this way, when checking whether t is contained in the message, we just do it for parent and not repeat for its descendants. Utilizing this idea, we keep two pivotal-token sets for a quadtree node: a cover set and a non-cover set. In the cover set, we keep the token appearing in all four children. For the no common tokens are kept in non-cover set, and they are not removed from children. We call such a quadtree Pivotal-Token Quadtree (PT-Quadtree).

For example, the subscriptions in Figure 1 can be constructed as a PT-Quadtree shown in Figure 4. Consider leaf nodes 2, 3, 4, 5 with subscriptions s_3, s_3, s_9, s_5. As the four nodes contain token "b", we select token "b" as their pivotal tokens. Obviously token "b" should be kept in their parent's cover set, i.e., node 1. For nodes 17, 18, 19, 20, we respectively select pivotal tokens d, b, b, b. As their pivotal-token token is not the same, the cover set of their parent (node 16) is ϕ and the non-cover set is $\{b, d\}$.

Space Complexity: Suppose the height of PT-Quadtree is H_q and there are N_q nodes in the quadtree. If we use the MX-CIF Quadtree [18], each subscription is stored only at the node corresponding to the smallest region that contains the subscription MBR, and the sum of lengths of all inverted lists is $|\mathcal{S}|$. To achieve high performance, we want to keep inverted lists only for leaf nodes, and the inverted-list size is $\alpha \times |\mathcal{S}|$, where α is a parameter and depends on datasets. There are at most $|\mathcal{S}|$ pivotal tokens in leaf nodes, and each token is at most stored H_q times. Thus the maximal size of pivotal-token sets is $\mathcal{O}(|\mathcal{S}| \times H_q)$. In practice, as many subscriptions share tokens, the size is much smaller. Thus the overall space complexity is $\mathcal{O}(N_q + \alpha \times |\mathcal{S}| + |\mathcal{S}| \times H_q)$.

Updates: For a new subscription, we insert into PT-Quadtree and select a pivotal token to insert into the non-cover set of nodes on the path from root to the leaf (if the non-cover set does not contain the token). For deletions, we can use a special mark to denote the deletions.

Algorithm 2: PT-QUADTREE $(\mathcal{S}, m)$

Input: $\mathcal{S}$: The subscription set; m: A message
Output: $\mathcal{R}$: Answers of m

```
1 begin
2   Build an PT-Quadtree with root r;
3   Initialize a candidate node set CN;
4   PT-QUADTREE-FILTER (r, m, CN) ;
5   R = PT-QUADTREE-VERIFY (m, CN);
6 end
```

Function PT-QUADTREE-FILTER(n, m, $\mathcal{CN}$)

Input: n: An PT-Quadtree node; m: A message
$\mathcal{CN}$: Leaf candidate node set

```
1 begin
2   if n ∩ m.R == ϕ then return CN;
3   if n.CSet ∩ m.T ≠ ϕ then
4     Traverse the subtree rooted at n, and add the
      leaf nodes that have overlaps with m.R to CN;
5     return CN ;
6   if n.NSet ∩ m.T ≠ ϕ then
7     for each child c of n do
8       PT-QUADTREE-FILTER (c, m, CN);
9 end
```

Function PT-QUADTREE-VERIFY(m, $\mathcal{CN}$)

Input: m: A message; $\mathcal{CN}$: leaf candidate node set
Output: $\mathcal{R}$: Answers of m

```
1 begin
2   for each node n ∈ CN do
3     for each subscription s ∈ I_n do
4       if s.T ⊆ m.T&s.R ∩ m.R ≠ ϕ then R ← s;
5 end
```

Figure 5: PT-Quadtree algorithm

4.2 Filtering Algorithms

For the PT-Quadtree, we introduce a filter-and-verification framework. In the filter step, we generate a set of leaf candidate nodes. Then we verify the subscriptions on the inverted lists of leaf candidate nodes in the verification step.

Step 1 - Filter: We traverse PT-Quadtree in a depth-first order. For each PT-Quadtree node n, let $n.CSet$ and $n.NSet$ respectively denote n's cover set and non-cover set.

(1) If $n \cap m.R = \phi$, we prune node n based on spatial constraint;

(2) If $n.CSet \cap m.T = \phi$ and $n.NSet \cap m.T = \phi$, we prune the node as it invalidates textual constraint;

(3) If $n \cap m.R \neq \phi$, $n.CSet \cap m.T = \phi$, and $n.NSet \cap m.T \neq \phi$, n is a candidate. We repeat the filter steps for its children.

(4) Otherwise $n \cap m.R \neq \phi$ and $n.CSet \cap m.T \neq \phi$, subscriptions on the inverted lists of n's leaf descendants share a common token with m. Thus we only check whether the descendants of n satisfy the spatial constraint. If yes, these leaf nodes are candidate nodes; otherwise we prune them.

Step 2 - Verification: Given a leaf candidate node, as subscriptions on its inverted list may not satisfy both constraints, we need a verification step to examine the subscriptions to get the final answers.

Next we devise an PT-Quadtree based algorithm as shown in Figure 5. First we construct a PT-Quadtree (line 2). Then for the message m, we call function PT-QUADTREE-FILTER to find candidate nodes from the root (line 4). Consider a candidate node n. PT-QUADTREE-FILTER first checks whether node n has an overlap with $m.R$. If not, we prune it (line 2); otherwise, we check whether it satisfies the textual constraint as follows. First if $n.CSet \cap m.T \neq \phi$, we need to access all of n's descendants (line 3, 4); otherwise, we check whether $n.NSet \cap m.T \neq \phi$ (line 6). If yes we need to visit n's children (line 8); otherwise, we prune the node. Iteratively we can get all leaf candidate nodes. Theorem 2 formalizes the correctness of our algorithm.

THEOREM 2. *The* PT-QUADTREE *algorithm satisfies completeness and correctness.*

For example, consider PT-Quadtree in Figure 4. For range message m_r in Figure 1. As the non-cover set of root contains token "d" which appears in $m_r.T$, we need to visit its children. Node 1 is pruned as its pivotal-token set has no overlap with the token set of m_r. We can also prune nodes 14, 19, and 20 as they have no spatial overlap with m_r. Thus the leaf candidate nodes are 7, 10, 12, 17. Next we verify the subscriptions s_2, s_{11}, and s_{10} on their inverted lists.

Time Complexity: We can check whether $m.T \cap n.CSet \neq \phi (m.T \cap n.NSet \neq \phi)$ using the cost model in Section 3.3. The time complexity is $\mathcal{O}\big(\min(\log N \times |n.CSet|, \log |n.CSet| \times N)\big)$, where N is the number of tokens in $m.T$. Suppose there are cand_Q leaf candidate nodes and the height of PT-Quadtree is H_q. The complexity of the filter step is

$$\mathsf{Cost}_Q^{\mathcal{F}} = \mathcal{O}\big(H_q \times \mathsf{cand}_Q \times \min(\log N \times Avg_{RT}, \log Avg_{RT} \times N)\big), \quad (3)$$

where Avg_{RT} is the average size of pivotal-token sets.

Note that each subscription of candidate nodes may not have overlap with the message. To find the final answers that satisfy both two constraint, we can use each token in $s.T$ to do a binary search on $m.T$ and the complexity is $\mathcal{O}(|s.T| \times \log N)$. Suppose there are cand_Q candidate leaf nodes, the inverted lists of them are $\mathcal{I}_1, \cdots, \mathcal{I}_{\mathsf{cand}_Q}$, and the average token size of the subscriptions is $\mathcal{S}_{avg}$. The time complexity of the verification step is

$$\mathsf{Cost}_Q^{\mathcal{V}} = \mathcal{O}\Big(\sum_{i=1}^{\mathsf{cand}_Q} |\mathcal{I}_i| \times \mathcal{S}_{avg} \times \log N\Big). \quad (4)$$

5. COST-BASED ALGORITHMS

In this section, we first compare MBRTrie and PT-Quadtree algorithms in Section 5.1. Then we propose a pruning technique in Section 5.2 and give a cost-based model in Section 5.3. Finally, we propose an algorithm in Section 5.4.

5.1 Comparison of MBRTrie and PT-Quadtree

In the verification step, MBRTrie only verifies the spatial constraint (Section 3.3) and PT-Quadtree needs to verify both spatial constraint and textual constraint (Section 4.2), thus MBRTrie has lower complexity than PT-Quadtree in verification. Next we consider the filter step of two methods.

Let $\mathsf{Cost}_T^{\mathcal{F}}$ and $\mathsf{Cost}_T^{\mathcal{V}}$ respectively denote the cost of the filter step and the verification step of MBRTrie which can be computed by Equations 1 and 2, and the cost of filter and verifications steps of PT-Quadtree are denoted by $\mathsf{Cost}_Q^{\mathcal{F}}$ and $\mathsf{Cost}_Q^{\mathcal{V}}$, respectively. They can be computed by Equations 3 and 4. We consider the following two cases.

Case 1: $\mathsf{Cost}_Q^{\mathcal{F}} > \mathsf{Cost}_T^{\mathcal{F}}$. This happens when the message's MBR is large, for example. $\mathsf{Cost}_Q^{\mathcal{F}}$ will be very large. Moreover MBRTrie is very efficient to verify the candidates in the verification step, thus we employ the MBRTrie algorithm.

Case 2: $\mathsf{Cost}_Q^{\mathcal{F}} \leq \mathsf{Cost}_T^{\mathcal{F}}$. In this case, we employ the PT-Quadtree algorithm in the filter step since PT-Quadtree is more efficient than MBRTrie in the filter step. For example, for a point message, as PT-Quadtree only accesses a single quadtree node in each level, $\mathsf{Cost}_Q^{\mathcal{F}}$ is very small. And in the verification step $\mathsf{Cost}_Q^{\mathcal{V}}$ is expensive, we have two strategies to verify candidates. The first verifies results as discussed in PT-Quadtree algorithm directly. The second utilizes MBRTrie algorithm to do further pruning by their inverted lists after generating leaf candidate nodes by PT-Quadtree. The detail of this method is described in the next subsection.

5.2 List Pruning

In the MBRTrie, subscriptions are sorted in the token-set order. Therefore subscriptions under node n must be in a range $[n_l, n_u]$ which means all the ID of subscriptions in the inverted list of n's leaf nodes are between n_l and n_u. Based on these ranges we can integrate MBRTrie and PT-Quadtree to filter a message. First we consider a point message.

For a leaf candidate node found by PT-Quadtree for the point message, there are two strategies to find answers from its inverted list $\mathcal{I}$. The first is to directly verify the subscriptions in $\mathcal{I}$. The second is to utilize MBRTrie and $\mathcal{I}$ to do further filtering as follows.

When we traverse MBRTrie, for a trie node n with subscription range $[n_l, n_u]$, we check whether $\mathcal{I}$ contains a subscription in $[n_l, n_u]$. We prune node n if there is no such subscription, as any subscription under it will not satisfy the spatial constraint. We call this technique *list pruning* which can prune many unnecessary subscriptions.

We utilize a binary search to check whether $\mathcal{I}$ has overlap with $[n_l, n_u]$. If $\mathcal{I}$ has no subscription within $[n_l, n_u]$, we prune node n. Otherwise, suppose the smallest ID not smaller than n_l is n_{min} and n_{max} is the largest ID that no larger than n_u. We generate a sublist $\mathcal{I}_n$ of $\mathcal{I}$ by selecting the subscriptions in $[n_{min}, n_{max}]$. Thus subscriptions under node n that satisfy the spatial constraint must be in $\mathcal{I}_n$. Next we use $\mathcal{I}_n$ to do pruning for n's children. Interactively we can find a set of leaf nodes $\mathcal{L}$. For each leaf node $l \in \mathcal{L}$, suppose its new list is $\mathcal{I}_l$. Each subscriptions in $\mathcal{I}_l$ must satisfy the textual constraint and the sum of number of subscriptions in the inverted list of leaf nodes in $\mathcal{L}$ is smaller than that of $\mathcal{I}$ ($\sum_{l \in \mathcal{L}} |\mathcal{I}_l| \leq |\mathcal{I}|$). Therefore we only verify subscriptions in $\mathcal{I}_l$ for spatial constraint and this complexity on new lists is smaller than that on the original list $\mathcal{I}$.

For example, consider PT-Quadtree in Figure 4. For point message m_p in Figure 1, we can generate the inverted list $\mathcal{I} = \{s_1, s_4, s_7, s_9\}$. Then we traverse MBRTrie in Figure 2. For the children of the root, we prune node 26 as $\mathcal{I}$ has no subscription in range ($[s_{11}, s_{12}]$) of node 26. For node 8 with range $[s_3, s_9]$, it generate a sublist $\mathcal{I}_n = \{s_4, s_7, s_9\}$. We can easily extend our method to support range messages.

5.3 Cost-based Model

Selecting an algorithm in the filter step: As discussing in Section 5.1. We select the filter algorithm based on the cost of $\mathsf{Cost}_Q^{\mathcal{F}}$ and $\mathsf{Cost}_T^{\mathcal{F}}$ as shown in Equation 1 and 3, re-

spectively. $\mathsf{Cost}_T^{\mathcal{F}}$ can be easily computed based on the MBRTrie structure and the message m. For $\mathsf{Cost}_Q^{\mathcal{F}}$, we only need to estimate cand_Q and other parameters can be gotten from the PT-Quadtree structure. For a point message $\mathsf{cand}_Q = 1$. Next we discuss how to estimate cand_Q for a range message. We can use the k-th level PT-Quadtree nodes to estimate the number of candidates. For each PT-Quadtree node n_i in the k-th level, suppose it has $|\texttt{leaves}(n_i)|$ leaf descendants. Obviously, the larger the spatial overlap $||n_i \cap m.R||$ is, the more leaf nodes have overlap with m under node n_i, where $||n_i \cap m.R||$ is the overlap area between node n_i and $m.R$. Thus we use the following equation to estimate the number,

$$\mathsf{cand}_Q = \sum_{n_i} |\texttt{leaves}(n_i)| \times \frac{||n_i \cap m.R||}{||n_i||}.$$

If $\mathsf{Cost}_Q^{\mathcal{F}} > \mathsf{Cost}_T^{\mathcal{F}}$, we use MBRTrie algorithm in both the filter and verification steps; otherwise, in the filter step we select PT-Quadtree algorithm. Next we discuss how to select an algorithm for the verification step in this situation.

Selecting an algorithm in the verification step: Given a trie node n in the i-th level and a list $\mathcal{I}_n$, suppose n matches the j-th token t_j in message token set T. As discussed in Section 3.3, we estimate the complexity to find n's descendants with tokens in $T_n = \{t_{j+1}, \cdots, t_N\}$ as below.

$$\mathcal{C}_F = \mathcal{O}\Big(\min\big(L_{i+1} \times \log|T_n| + L_{i+2} \times \frac{|T_n|}{D} \times \log|T_n|, \log L_{i+1} \times |T_n| + \log L_{i+2} \times \frac{|T_n|}{D} \times |T_n|\big)\Big).$$

For n's leaf descendants, we verify subscriptions on their lists with the complexity $\mathcal{O}(\mathcal{I}_n)$. As we need to use the range of n to probe $\mathcal{I}_n$ with complexity $\mathcal{O}(\log|\mathcal{I}_n|)$, the total complexity of the list-pruning technique is

$$\mathcal{C}_L = \mathcal{C}_F + \log|\mathcal{I}_n| + \mathcal{I}_n. \quad (5)$$

The time complexity to directly verify the list is

$$\mathcal{C}_V = \mathcal{O}(|\mathcal{I}_n| \times \mathcal{S}_{avg} \times \log|T_n|), \quad (6)$$

where $\mathcal{S}_{avg}$ is the average token number of subscriptions.

Initially we consider the root node and the inverted list gotten by PT-Quadtree. If $\mathcal{C}_L > \mathcal{C}_V$, we directly verify the list; otherwise we traverse MBRTrie using list-pruning technique. Iteratively, we always select a better verification model for each accessed node.

For multiple inverted lists, $\mathcal{C}_L = \mathcal{O}(\mathcal{C}_F + \sum_{\mathcal{I}\in\Phi_{\mathcal{I}}}(\log|\mathcal{I}| + \mathcal{I}))$ and $\mathcal{C}_V = \mathcal{O}(\sum_{\mathcal{I}\in\Phi_{\mathcal{I}}}|\mathcal{I}| \times \mathcal{S}_{avg} \times \log|T_n|)$, where $\Phi_{\mathcal{I}}$ is the set of inverted lists generated by PT-Quadtree.

5.4 LFilter Algorithm

In this section, we devise an efficient location-based filtering algorithm, LFilter, asillustrated in Figure 6.

In LFilter, we first construct an PT-Quadtree (line 2) and an MBRTrie (line 3). If $\mathsf{Cost}_Q^{\mathcal{F}} > \mathsf{Cost}_T^{\mathcal{F}}$, LFilter calls MBRTrie algorithm (line 5); otherwise it calls function PT-Quadtree-Filter to generate the candidate nodes based on PT-Quadtree (line 7). Then based on the inverted-list set $\Phi_{\mathcal{I}}$, LFilter calls function HybridPrune to filter the message using a cost model (line 9).

For each trie node n, if n is a leaf node, we check whether subscriptions on its inverted list satisfy spatial constraint (line 3). Next we select a better verification strategy for the inverted lists. If $\mathcal{C}_L > \mathcal{C}_V$, we directly verify whether the subscriptions on the inverted lists on it satisfy both two constraints (line 6); otherwise we use *list pruning* technique on

Algorithm 3: LFilter $(\mathcal{S}, m)$

Input: $\mathcal{S}$: The subscription set; m: A message
Output: $\mathcal{R}$: Answers of m

```
1  begin
2      Build an PT-Quadtree with root r_q;
3      Build an MBRTrie with root r_t ;
4      if Cost_Q^F > Cost_T^F then
5          R = MBRTrie (S, m) ;
6      else
7          PT-Quadtree-Filter (r_q, m, CN) ;
8          Compute inverted-list set Φ_I of nodes in CN;
9          HybridPrune (r_t, m, Φ_I, R) ;
10 end
```

Function HybridPrune$(n, T_n, \Phi_{\mathcal{I}}, \mathcal{R})$

Input: n: A trie node; m: A message
$\Phi_{\mathcal{I}}$: A set of inverted lists; $\mathcal{R}$: Answers of m

```
1  begin
2      if n is a leaf node then
3          R∪ = MBRTrie-Verify (m, {n});
4      if C_L > C_V then
5          for I ∈ Φ_I do
6              R∪ = PT-Quadtree-Verify (m, I);
7              //verify the subscriptions on the list I
8      else
9          P = SelectPair (n, T_n, m) ;
10         for each pair ⟨c, T_c⟩ ∈ P do
11             Φ'_I = ListPruning (c, Φ_I); //∀I ∈ Φ_I,
               compute sublist of I in range [c_l, c_u], I_c
12             HybridPrune (c, T_c, Φ'_I, R);
13 end
```

Figure 6: LFilter algorithm

n's children (line 9), update the inverted lists using its children' ranges (line 11), and recursively check whether access its children (line 12). Iteratively we can find the answers and Theorem 3 guarantees the correctness and completeness.

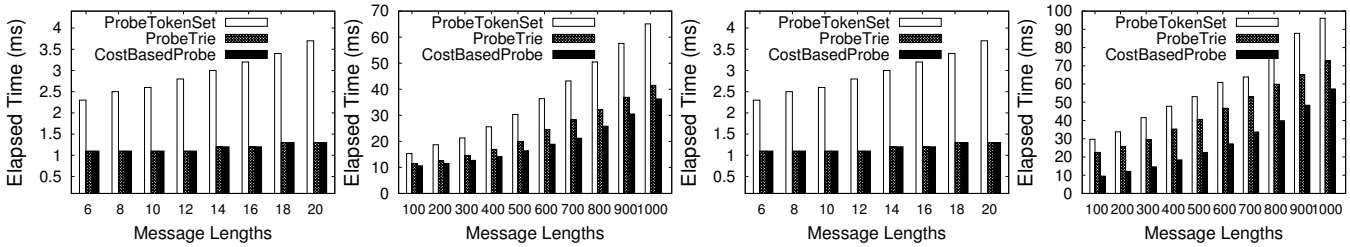
Theorem 3. *The* LFilter *algorithm satisfies complexness and correctness.*

6. EXPERIMENTAL STUDY

We compared with state-of-the-art location-aware publish/subscribe method R^{t++}-Tree [14] and location-aware keyword search method IRTree [5][5].

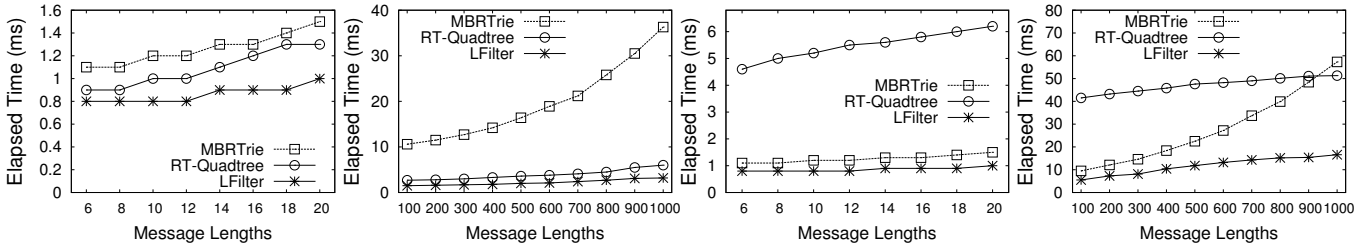
We used two datasets. The first one was a real dataset Twitter. We collected 60 million tweets and selected 10 million tweets with region information as subscriptions and used others as messages. Each subscription contained an MBR and had 1-5 frequent tokens selected from tweets. The average token number of subscriptions was 3. The second dataset was 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

[5]We extended it to support our problem by checking whether IR-tree nodes can be pruned with our filters.



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

Figure 7: Evaluating probing strategies in MBRTrie algorithm on the Twitter dataset (using idf order)



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

Figure 8: Comparison of MBRTrie, PT-Quadtree and LFilter on the Twitter dataset

and height. Each subscription was generated by selecting an MBR and 1-6 tokens from DBLP. Each message was generated by selecting an MBR and a publication.

To evaluate different algorithms, we generated four groups of messages for both datasets by considering the number of tokens and a point location or an region. For the length of messages, the short and long messages contained 6-20 and 100-1000 tokens, respectively. And each group contained 10, 000 messages. We computed the average elapsed time. Table 1 showed datasets and index sizes, where subscription length represents the number of tokens in a subscription. Due to space constraints, the results on USA dataset were only shown in Section 6.3 to compare with existing methods.

All the algorithms were implemented in C++. All the experiments were run on a machine running Ubuntu with a 3.0GHz CPU and 8 GB memory.

Table 1: Dataset statistics.

	Twitter	USA
Subscription number	10 million	10 million
Subscription length	1-5	1-6
Avg Subscription length	3	3.5
Subscription size	542 MB	654 MB
MBRTrie size	1.33 GB	1.45 GB
PT-Quadtree size	1.28 GB	1.37 GB

6.1 Evaluating MBRTrie Algorithm

We evaluated the MBRTrie algorithm by varying different probe strategies: ProbeTokenSet, ProbeTrie, and CostBasedProbe, as discussed in Section 3.3. Figure 7 shows the results. We can see that CostBasedProbe achieved the best performance as it adaptively selected a better probing strategy. For upper-level nodes which had large numbers of children, CostBasedProbe used messages to probe trie structures; when the nodes in the lower-level had small number, it used the trie nodes to probe messages. For instance, in Figure 7(d), for messages with 1000 tokens, ProbeTrie took 98 milliseconds, ProbeTokenSet took 72 milliseconds, and CostBasedProbe only took 56 milliseconds.

6.2 Comparison of MBRTrie, PT-Quadtree, LFilter

We compared our three algorithms, MBRTrie, PT-Quadtree, and LFilter by varying the message lengths. The results are shown in Figure 8. LFilter significantly outperformed other two algorithms as it used a cost-based model to select a better strategy to deal with any types of messages. PT-Quadtree was better than MBRTrie for point messages, since it only accessed one node in each level (see Figure 8(a) and 8(b)). However, for range messages, MBRTrie achieved higher performance than PT-Quadtree, as PT-Quadtree was very expensive in the verification step (see Figures 8(c) and 8(d)). For example, in Figure 8(d), for messages with 1000 tokens, LFilter took 16 milliseconds and MBRTrie and PT-Quadtree took more than 50 milliseconds.

6.3 Comparison with Existing Methods

We compared our LFilter with existing methods, R^{t++}-Tree [14] and IRTree [5]. Figures 9 and 10 showed the results on Twitter and USA respectively.

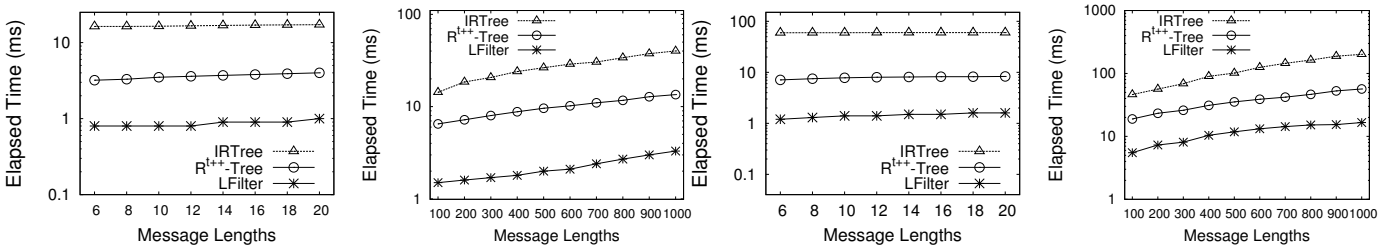
We can see that LFilter outperformed R^{t++}-Tree, which in turn was better than IRTree. The reason is that the two existing methods had low pruning power on the textual part. For each R-tree node, if the node contained a token in the message, their method had to visit its children. Thus it accessed huge number of unnecessary nodes. So IRTree and R^{t++}-Tree were not efficient for the location-aware filtering problem. Our algorithm always achieved the highest performance for any types of messages utilizing its cost-based model to select a better filtering strategy for both filtering and verification. For example, in Figure 9(b), IRTree took more than 15 milliseconds, R^{t++}-Tree decreased to 10 milliseconds, and LFilter only took 2-4 milliseconds.

6.4 Scalability

Figure 11 showed the results of the scalability of our method by varying the numbers of subscriptions. We can see that our method scaled very well, and with increasing the numbers of subscriptions, the elapsed time increased almost linearly. For example, for messages with 600 tokens, the performance was 6 milliseconds for 2 million subscriptions and the elapsed time for 10 million subscriptions was only 16 milliseconds. This is because even if the number of subscriptions increased, our index structures can pruned a large number of unnecessary subscriptions.

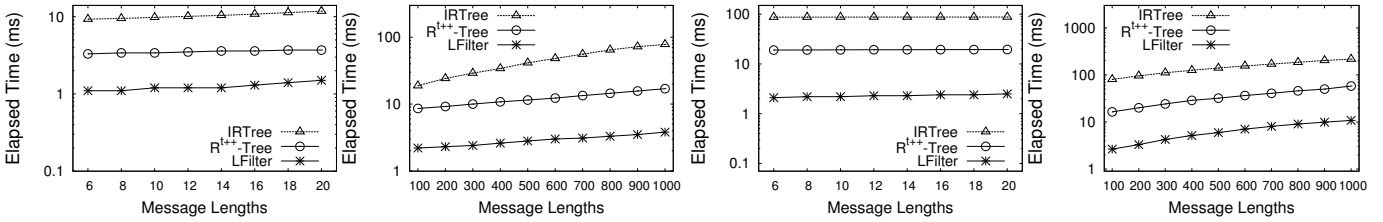
7. CONCLUSION

In this paper we have studied the location-aware publish/subscribe problem. We propose two new index struc-



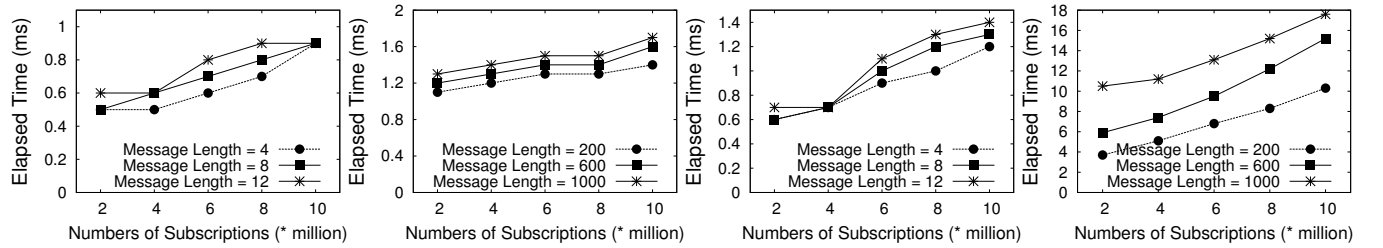
(a) Short Point Messages (b) Long Point Messages (c) Short Range Messages (d) Long Range Messages

Figure 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

Figure 10: Comparison with existing studies on the USA dataset



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

Figure 11: Scalability on the Twitter dataset

tures MBRTrie and PT-Quadtree to address this problem. MBRTrie is efficient for the messages with a small number of tokens and PT-Quadtree achieves high performance for messages with small regions. We also develop a cost-based model LFilter to select the best filtering strategy to achieve high performance for any types of messages. Experimental results show that our method achieves high performance and significantly outperforms existing approaches.

Acknowledgement. This work was supported by 973 Program of China (2015CB358700), and NSF of China (61272090, 61373024), Beijing Higher Education Young Elite Teacher Project (YETP0105), Tencent, SAP, Huawei, the "NExT Research Center" funded by MDA, Singapore (WBS:R-252-300-001-490), and FDCT/106/2012/A3.

8. REFERENCES

[1] G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *J. Comput. Syst. Sci.*, 21(1):136–153, 1980.

[2] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *PVLDB*, 3(1):373–384, 2010.

[3] L. Chen, G. Cong, and X. Cao. An efficient query indexing mechanism for filtering geo-textual data. In *SIGMOD Conference*, pages 749–760, 2013.

[4] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD Conference*, pages 277–288, 2006.

[5] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2009.

[6] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe. In *SIGMOD*, pages 115–126, 2001.

[7] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, 2008.

[8] P. W. Foltz and S. T. Dumais. Personalized information delivery: An analysis of information filtering methods. *Commun. ACM*, 35(12):51–60, 1992.

[9] L. Guo, D. Zhang, G. Li, K. lee Tan, and Z. Bao. Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In *SIGMOD*, 2015.

[10] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *SSDBM*, 2007.

[11] H. Hu, G. Li, Y. Liu, J. Feng, and K.-L. Tan. A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In *ICDE*, 2015.

[12] P. Jin, H. Chen, S. Lin, X. Zhao, and L. Yue. Hybrid index structures for temporal-textual web search. In *APWeb*, pages 271–277, 2011.

[13] G. Li, J. Feng, and J. Xu. DESKS: direction-aware spatial keyword search. In *ICDE*, pages 474–485, 2012.

[14] G. Li, Y. Wang, T. Wang, and J. Feng. Location-aware publish/subscribe. In *KDD*, pages 802–810, 2013.

[15] X. Liu, L. Chen, and C. Wan. LINQ: A framework for location-aware indexing and query processing. *IEEE Trans. Knowl. Data Eng.*, 27(5):1288–1300, 2015.

[16] J. Lu, Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD Conference*, pages 349–360, 2011.

[17] S. B. Roy and K. Chakrabarti. Location-aware type ahead search on spatial databases: semantics and efficiency. In *SIGMOD*, pages 361–372, 2011.

[18] H. Samet. *Foundations of Multidimensional and Metric Data Structure*. 2006.

[19] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. Ap-tree: Efficiently support continuous spatial-keyword queries over stream. In *ICDE 2015*, pages 1107–1118, 2015.

[20] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552, 2011.

[21] T. W. Yan and H. Garcia-Molina. Index structures for information filtering under the vector space model. In *ICDE*, pages 337–347, 1994.

[22] T. W. Yan and H. Garcia-Molina. The sift information dissemination system. *ACM Trans. Database Syst.*, 24(4):529–565, 1999.

[23] B. Yao, F. Li, M. Hadjieleftheriou, and K. Hou. Approximate string search in spatial databases. In *ICDE*, 2010.

[24] M. Yu, G. Li, T. Wang, J. Feng, and Z. Gong. Efficient filtering algorithms for location-aware publish/subscribe. *IEEE Trans. Knowl. Data Eng.*, 27(4):950–963, 2015.

[25] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, 2005.