SEAL: Spatio-Textual Similarity Search

Ju Fan  Guoliang Li  Lizhu Zhou  Shanshan Chen  Jun Hu
Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China.
fan-j07@mails.tsinghua.edu.cn, liguoliang@tsinghua.edu.cn, dcszlz@tsinghua.edu.cn, aarifc33@gmail.com, j-hu08@mails.tsinghua.edu.cn

ABSTRACT
Location-based services (LBS) have become more and more ubiquitous recently. Existing methods focus on finding relevant points-of-interest (POIs) based on users’ locations and query keywords. Nowadays, modern LBS applications generate a new kind of spatio-textual data, regions-of-interest (ROIs), containing region-based spatial information and textual description, e.g., mobile user profiles with active regions and interest tags. To satisfy search requirements on ROIs, we study a new research problem, called spatio-textual similarity search: Given a set of ROIs and a query ROI, we find the similar ROIs by considering spatial overlap and textual similarity. Spatio-textual similarity search has many important applications, e.g., social marketing in location-aware social networks. It calls for an efficient search method to support large scales of spatio-textual data in LBS systems. To this end, we introduce a filter-and-verification framework to compute the answers. In the filter step, we generate signatures for the ROIs and the query, and utilize the signatures to generate candidates whose signatures are similar to that of the query. In the verification step, we verify the candidates and identify the final answers. To achieve high performance, we generate effective high-quality signatures, and devise efficient filtering algorithms as well as pruning techniques. Experimental results on real and synthetic datasets show that our method achieves high performance.

1. INTRODUCTION
Nowadays, as mobile devices (e.g., smartphones) with built-in global position systems (GPS) become more and more popular, location-based services (LBS) have been widely accepted by mobile users and attracted significant attention from both the academic and industrial community. Many location-based services, such as Foursquare and Facebook Places, bring unique location-aware experiences to users.

Existing LBS systems employ a spatial keyword search approach to provide LBS services [9,7], which, given a set of points of interest (POIs), and a user query with location and keywords, finds all relevant POIs. For example, if a user wants to find the coffee shops nearby, she can issue a keyword query “coffee shop” to an LBS system, which returns the relevant coffee shops by considering the user’s location and query keywords.

Recently, many modern LBS applications generate a new kind of spatio-textual data, regions-of-interest (ROIs), containing region-based spatial information and textual description. For example, in Facebook Places, mobile users have profiles consisting of active regions and interest tags. In wildlife monitoring, wild species with their habitats and descriptive features can be modeled by ROIs.

To satisfy search requirements on ROIs, we introduce a new research problem, called spatio-textual similarity search in this paper: Given a set of ROIs and a query ROI, we aim to find the ROIs which are similar to the query by considering spatial overlap and textual similarity.

Spatio-textual similarity search can satisfy users’ information needs in various real applications. The first one is location-based social marketing using Facebook Places. As mentioned above, in Facebook Places, mobile users have profiles that can be modeled by ROIs. For example, consider some users in Manhattan who are interested in tea and coffee. A coffee shop (e.g., starbucks) can utilize user profiles in Facebook to provide location-specific advertisements to the potential customers who not only are interested in its products (e.g., {starbucks, mocha, coffee}) but also have region-based spatial overlap with its service area. Another example is friend recommendation in location-aware social networks, e.g., Facebook, Foursquare and Twitter. Spatio-textual similarity search helps mobile users find potential friends with common interests (e.g., playing basketball) and overlap regions (e.g., Brooklyn), and thus facilitates users to form various kinds of circles with the same interests, such as sport games, shopping, and fans’ activities. Spatio-textual similarity search can also support other applications, e.g., wildlife protection. Wild species have their habitats (e.g., Yellowstone National Park for grizzly bears) and features (e.g., mammal, omnivore, etc.). A zoologist can issue a query to find all wild species having certain features (e.g., mammal) and inhabiting in a specific region (e.g., Idaho).

In this paper, we formalize the problem of spatio-textual similarity search, and study the research challenges that naturally arise in this problem. A challenge is how to evaluate...
the similarity between two ROIs. Another challenge is how to achieve high search efficiency as LBS systems are required to support millions of users and respond to queries in milliseconds. Given a query ROI, there may be a huge amount of ROIs having significant overlaps with the query, thus it is rather expensive to find similar answers. Take the real dataset Twitter in our experiments (See Section 1) as an example. We used a set of query regions in an average area of 0.4 square kilometers. Even for one of the small query regions, there were, on average, 8000 ROIs overlapping with it. Moreover, similarity search needs to consider both spatial and textual similarities. To address these challenges, we propose an efficient Spatio-Textual Similarity search method, called SEAL-Search. We combine spatial similarity functions and textual similarity functions to quantify the similarity between two ROIs. To provide high performance, we introduce a filter-and-verification framework to compute the answers. In the filter step, our method generates signatures for spatio-textual objects and queries, and utilizes the signatures to generate candidates whose signatures are similar to those of the queries. In the verification step, it verifies the candidates and identifies the final answers. We develop effective techniques to generate signatures and devise efficient filtering algorithms to prune dissimilar objects.

To summarize, we make the following contributions:

- To the best of our knowledge, we are the first to study spatio-textual similarity search on ROIs. We propose a filter-and-verification framework and signature-based filtering algorithms to address this problem.
- For effective spatial pruning, we devise grid-based signatures and develop threshold-aware pruning techniques.
- To utilize spatial and textual pruning simultaneously, we judiciously select high-quality signatures and devise efficient hybrid filtering algorithms.
- We have conducted extensive experiments on real and synthetic datasets. Experimental results show that our algorithms achieve high performance.

The paper is organized as follows. The problem formulation and related works are presented in Section 2. We introduce a signature-based method in Section 3. We develop grid-based filtering algorithms in Section 4 and hybrid filtering algorithms in Section 5. Experimental results are provided in Section 6. Finally, we conclude the paper and discuss the future work in Section 7.

## 2. PRELIMINARIES

### 2.1 Problem Formulation

**Data Model.** Our work focuses on supporting similarity search for a set of spatio-textual ROI objects (or objects for simplicity), \( \mathcal{O} = \{o_1, o_2, \ldots, o_n\} \). Each object \( o \in \mathcal{O} \) consists of spatial information \( o.R \) and textual information \( o.T \), denoted by \( o = (R, T) \). The spatial information \( o.R \) is a region. We use the well-known minimum bounding rectangle (MBR) to represent region \( o.R \) through the bottom-left point and top-right point of the MBR. The textual description \( o.T \) is a set of tokens, i.e., \( \{t_1, t_2, \ldots, t_{|o.T|}\} \), where each token \( t \in o.T \) is associated with a weight \( w(t) \) to capture its importance. Figure 1 illustrates an example of seven objects, each of which has several tokens and a region.

**Query Model.** Our paper considers a spatio-textual similarity search query \( q \) that also consists of a region \( q.R \) and a set of tokens \( q.T \). Given a set of objects \( \mathcal{O}\) and query \( q\), the answers of query \( q\) are a set of objects \( \mathcal{A} \subseteq \mathcal{O} \) similar to \( q\), i.e., for \( o \in \mathcal{A} \):

1. **The Spatial Similarity** \( \text{sim}_s(q, o) \geq \tau_s \),
2. **The Textual Similarity** \( \text{sim}_t(q, o) \geq \tau_t \),

where \( \tau_s \) and \( \tau_t \) are respectively spatial and textual similarity thresholds satisfying \( 0 \leq \tau_s, \tau_t \leq 1 \). Notice that we use the two thresholds to allow users to determine the spatial relevance and textual relevance in a more flexible way.

We quantify the spatial similarity based on the overlap of regions. Given two regions \( q.R \) and \( o.R \), their overlap is formally defined as the area of the intersecting region of \( q.R \) and \( o.R \), denoted by \( |q.R \cap o.R| \). Note that we use operator \( \cdot \) to represent both the cardinality of a set and the area of a region, if there is no ambiguity. Based on the overlap, we consider the Jaccard similarity in this paper.

**Definition 1** (Spatial Similarity). The spatial Jaccard similarity for two regions \( q.R \) and \( o.R \) is defined as

\[
\text{sim}_s(q, o) = \frac{|q.R \cap o.R|}{|q.R \cup o.R|},
\]

where \( |q.R \cup o.R| = |q.R| + |o.R| - |q.R \cap o.R| \).

For example, the overlap of regions \( o_1.R \) and \( q.R \) in Figure 1 is \( |q.R \cap o_1.R| = 1000 \) and \( |q.R \cup o_1.R| = 4400 \). Thus the spatial similarity is \( \text{sim}_s(q, o_1) = 1000/4400 = 0.23 \).

Note that our method can be easily extended to other overlap-based functions, such as Dice Similarity.

On the other hand, textual similarity measures the similarity between two token sets \( q.T \) and \( o.T \). Many token-set based similarity functions have been studied in string similarity join/search \([3, 8, 22]\), such as Jaccard similarity, Dice similarity, Cosine similarity, etc. In this paper, we take the weighted Jaccard similarity as an example.

**Definition 2** (Textual Similarity). The textual similarity \( \text{sim}_t(q, o) \) between \( q.T \) and \( o.T \) is defined as the weighted Jaccard coefficient of the two token sets, i.e.,

\[
\text{sim}_t(q, o) = \frac{\sum_{t \in q.T \cap o.T} w(t)}{\sum_{t \in q.T \cup o.T} w(t)},
\]

where \( w(t) \) is the weight of token \( t \).

In this paper we use the inverted document frequency (denoted by \( \text{idf} \)) as token weight, i.e., \( w(t) = \ln \frac{|\mathcal{O}|}{\text{count}(t, \mathcal{O})} \), where
Consider a set of spatio-textual objects and a spatio-textual similarity search query \( q \). Figure 1 shows five distinct tokens (i.e., \( t_1 \sim t_5 \)) with their weights. Using the weights, we compute textual similarity \( \text{sim}(q, o_1) = \left( w(t_1)+w(t_2) \right) / \left( w(t_1)+w(t_2)+w(t_3) \right) = 0.58 \).

Based on the above-mentioned notations, we formalize the spatio-textual similarity search problem.

**Definition 3 (Spatio-Textual Similarity Search).** Consider a set of spatio-textual objects \( O = \{ o_1, o_2, \ldots, o_\sigma \} \) and a spatio-textual similarity search query \( q = (R, T, \tau_6, \tau_7) \). It returns the objects \( A \subseteq O \) such that spatial similarity \( \text{sim}_R(q, o) \geq \tau_6 \) and textual similarity \( \text{sim}_T(q, o) \geq \tau_7 \), i.e., \( A = \{ o \mid o \in O, \text{sim}_R(q, o) \geq \tau_6, \text{sim}_T(q, o) \geq \tau_7 \} \).

**Example 1.** Consider the objects \( \{ o_1, o_2, \ldots, o_\sigma \} \) and a query \( q = (R_g, \{ t_1, t_2, t_3 \}, 0.25, 0.3) \) in Figure 1. Object \( o_2 = (R_2, \{ t_1, t_2, t_3 \}) \) is an answer since it satisfies \( \text{sim}_R \geq 0.32 \geq \tau_6(0.25) \) and \( \text{sim}_T = 1 \geq \tau_7(0.3) \). In contrast, object \( o_1 = (R_1, \{ t_1, t_2 \}) \) is not an answer due to \( \text{sim}_R = 0.23 \geq \tau_6 \), although satisfying \( \text{sim}_T = 0.58 \geq \tau_7 \). Considering all the objects in \( O \), we obtain the answer of \( q \), \( A = \{ o_2 \} \).

### 2.2 Related Work

**Spatial Keyword Search:** There are many studies on spatial keyword search \([23, 1, 11, 6, 23, 22, 3, 20, 24, 22, 3, 20, 24, 22, 3, 20, 24, 22, 3, 20] \). One problem is \( \text{IMX} \) based keyword search, which, given a query consisting of a location and a set of keywords, finds top-\( k \) relevant POIs by considering distance and textual relevance. Felipe et al. \([4]\) combined inverted files into R-tree, and Cong et al. \([1]\) extended inverted files and R-tree. Another problem is region-based keyword search, which, given a query consisting of a region and a set of keywords, finds the relevant POIs relevant to the keywords in the region. The methods addressing the problem also employed the R-tree index, and integrated inverted lists of keywords into R-tree nodes \([23, 1, 11, 6] \).

Our spatio-textual similarity search problem is substantially different from the above-mentioned problems. The underlying data is a set of spatio-textual objects consisting of regions and tokens (i.e., ROIs), rather than POIs. Moreover, the query model is different, and we focus on spatio-textual similarity between objects and queries and devise efficient filtering algorithms for similarity search.

**String Similarity Search/Join:** The problem of string similarity search/join has been extensively studied \([10, 1, 2, 11, 6, 23, 1, 11, 6, 23, 1, 11, 6] \). Given a set of strings and a query string, string similarity search finds the strings whose similarities to the query are not smaller than a threshold \( \tau \). Existing studies employed various functions, e.g., edit distance and Jaccard similarity, to quantify the similarity. To improve the performance, Chaudhuri et al. \([4]\) proposed a prefix filtering framework, and Bayardo et al. \([2]\) employed this framework to support Jaccard or Cosine similarity functions.

The basic idea of prefix filtering is to estimate a similarity upper bound of two sets using their subsets. Consider a string object \( o \) and a query string \( q \). The prefix filtering framework first maps both strings to sets, denoted by \( S(o) \) and \( S(q) \), and transforms various similarity functions to the overlap similarity on sets. More formally, if \( \text{sim}(q, o) \geq \tau \), then the sets satisfy \( |S(q)\cap S(o)| \geq c \), where \( c \) is a threshold deduced from \( \tau \). Then, the framework fixes a global order on the elements of all sets, and sorts the elements in each set based on the global order. Let \( S^p(o) \) denote the prefix...
3. THE SEAL METHOD

In this section, we first introduce a filter-and-verification framework in Section 3.1 and then present a textual-based filtering algorithm in Section 3.2.

3.1 A Filter-and-Verification Framework

To answer a spatio-textual similarity search query efficiently, we want to prune dissimilar objects and only visit a small amount of objects that may be similar to the query. To this end, we propose a filter-and-verification framework.

Step 1 - Filter: We prune a large amount of objects which cannot be similar to query \( q \), and find a candidate set \( C \), which is a superset of the answer set \( A \).

Step 2 - Verification: We verify the candidates generated in the filter step by checking whether spatial and textual similarities of each candidate are respectively not smaller than thresholds \( \tau_o \) and \( \tau_t \), and return the answer set \( A \).

In this paper, we focus on the filter step and propose efficient signature-based filtering algorithms. Consider a spatio-textual object \( o \in O \) and a query \( q \). We denote their signatures as \( S(o) \) and \( S(q) \), where each \( s \in S(o) \) (or \( S(q) \)) is called a signature element (or element for simplicity). A signature method must satisfy the following property:

\[
\text{o is similar to q only if } S(o) \text{ and } S(q) \text{ are similar.}
\]

Specifically, given a signature similarity function \( \text{sim}(\cdot, \cdot) \) and a threshold \( c \) deduced from thresholds \( \tau_o \) and \( \tau_t \), the object \( o \) is similar to \( q \) only if \( \text{sim}(S(o), S(q)) \geq c \).

A naïve method for generating candidates enumerates the signature of each object \( o \) and checks if \( \text{sim}(S(o), S(q)) \geq c \). If so, we add \( o \) into candidate set \( C \). However, this method is rather expensive and we want to build indexes to support efficient filtering, which will be discussed later.

Next we introduce our algorithm SEALSig and Figure 3 illustrates the pseudo-code. We first generate the signatures for objects in \( O \) and build an index on top of the signatures (line 2). Then for a query \( q \), we use the index to find its candidate set \( C \) (line 3). Finally, we verify the candidates in \( C \) and return the answer set \( A \) (line 4).

In this paper we focus on generating effective signatures, building indexes and devising efficient filtering algorithms.

3.2 A Textual-Based Filtering Algorithm

We introduce a textual-based filtering algorithm.

Textual Signatures. Since object \( o \) similar to query \( q \) must satisfy \( \text{sim}_T(q, o) \geq \tau_t \), we can use the tokens in \( o \) as its textual signature, i.e., \( S_T(o) = o.T \), where each token \( t \in o.T \) is a signature element. Similarly, we can also generate a textual signature for \( q \), i.e., \( S_T(q) = Q.T \).

Algorithm 1: SEALSig \((O, q)\)

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O ): An object set; ( q ): A query</td>
<td>( A ): Answers of ( q )</td>
</tr>
<tr>
<td>1 begin</td>
<td>2 Generate signatures for ( O ) and build index ( I );</td>
</tr>
<tr>
<td>3 ( C = \text{Sig-Filter}(q, I) );</td>
<td>4 ( A = \text{Sig-Verify}(q, C) );</td>
</tr>
<tr>
<td>5 end</td>
<td></td>
</tr>
</tbody>
</table>

Function \text{Sig-Filter}(q, I)

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q ): A query; ( I ): An inverted index</td>
<td>( C ): Candidate objects</td>
</tr>
<tr>
<td>1 begin</td>
<td>2 Initialize a candidate set ( C \leftarrow \emptyset );</td>
</tr>
<tr>
<td>3 Generate signature, ( S(q) \leftarrow \text{GENSig}(q) );</td>
<td>4 Compute signature similarity threshold ( c );</td>
</tr>
<tr>
<td>5 for each element ( s ) in ( S(q) ) do</td>
<td>6 Obtain objects in inverted list ( I(s) );</td>
</tr>
<tr>
<td>7 Merge objects with ( \text{sim}(S(q), S(o)) \geq c ) to ( C );</td>
<td></td>
</tr>
<tr>
<td>8 end</td>
<td></td>
</tr>
</tbody>
</table>

Function \text{Sig-Verify}(q, C)

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q ): A query; ( C ): A set of candidate objects</td>
<td>( A ): Answers of ( q )</td>
</tr>
<tr>
<td>1 begin</td>
<td>2 for each object ( o \in C ) do</td>
</tr>
<tr>
<td>3 if ( \text{sim}_R(q, o) \geq \tau_R ) and ( \text{sim}_T(q, o) \geq \tau_T ) then</td>
<td>4 ( A \leftarrow A \cup {o} );</td>
</tr>
<tr>
<td>4 end</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: A Filter-and-Verification Framework

Then, we define the similarity between signatures \( S_T(o) \) and \( S_T(q) \) as the weight summation of their common tokens, i.e., \( \text{sim}(S_T(q), S_T(o)) = \sum_{t \in S_T(q) \cap S_T(o)} w(t) \), and threshold \( c_t = \tau_t \cdot \sum_{t \in S_T(q) \cap S_T(o)} w(t) \). It is easy to prove that \( \text{sim}_T(q, o) \geq \tau_t \) only if \( \sum_{t \in S_T(q) \cap S_T(o)} w(t) \geq c_T \). For example, we can respectively generate textual signatures for \( o_1 \) and \( q \) as \( S_T(o_1) = \{t_1, t_2\} \) and \( S_T(q) = \{t_1, t_2, t_3\} \). Given \( \tau_T = 0.3 \), the threshold \( c_t \) can be computed as 0.57. Obviously, since \( \text{sim}_T(q, o_1) \geq \tau_T \), we have \( \sum_{t \in S_T(q) \cap S_T(o_1)} w(t) \geq c_T \).

Indexing Structures. To avoid enumerating every object \( o \in O \) for computing \( \text{sim}(S(q), S(o)) \), we build an inverted index on top of the signatures. Formally, an inverted index \( I \) consists of a set of inverted lists, each of which maps an element \( s \) to the objects containing the element, denoted by \( I(s) \). Figure 4 provides the inverted index of the objects in Figure 3. For example, since the signatures of object \( o_3 \) and \( o_8 \) contain element \( t_4 \), the inverted list of \( t_4 \) is \( \{o_3, o_8\} \). In addition, since \( o_3 \) has textual signature \( S_T(o_3) = \{t_1, t_3, t_2\} \), the object is contained in the inverted lists of \( t_1, t_3 \) and \( t_2 \).

Filtering Algorithm. Given a query \( q \) with thresholds \( \tau_o \) and \( \tau_t \), the filtering algorithm \( \text{SIG-FILTER} \) in Figure 3 is utilized to filter dissimilar objects and find the candidates with signatures similar to \( S(q) \), i.e., \( C = \{o \mid o \in O, \text{sim}(S(q), S(o)) \geq c\} \). Specifically, the algorithm generates a signature for \( q \), i.e., \( S(q) \leftarrow \text{GENSig}(q) \), and computes threshold \( c \). For each element \( s \in S(q) \), \( \text{SIG-FILTER} \) probes inverted list \( I(s) \) from the inverted index \( I \), and merges the objects in \( I(s) \) satisfying \( \text{sim}(S(q), S(o)) \geq c \) to \( C \).
4. GRID-BASED FILTERING ALGORITHM

In this section, we propose a grid-based filtering algorithm. We first define the grid-based signature in Section 4.1 and then devise a threshold-aware pruning technique in Section 4.2.

Finally, we discuss how to select grid granularity to achieve better performance in Section 4.3.

4.1 Grid-Based Signatures

Different from textual signatures, as the spatial information has no inherent elements (e.g., tokens), it is not straightforward to generate spatial signatures for objects in \( \mathcal{O} \) and query \( q \). To address this challenge, we propose to partition the entire space \( \mathcal{R} \) which is the MBR of the regions of all objects in \( \mathcal{O} \), and generate a set of grids. Then, for an object \( o \), we use the grids intersecting with region \( o.R \) as its spatial signature. More formally, let \( G = \{g_1, g_2, \ldots, g(|G|)\} \) be a set of grids in space \( \mathcal{R} \), where each grid is also an MBR, and the grids have the following properties:

1) **Completeness**: All grids cover the space, i.e., \( \bigcup_{g \in G} g = \mathcal{R} \).
2) **Disjointness**: Each pair of different grids is disjoinable, i.e., \( \forall i, j, \text{if } i \neq j, g_i \cap g_j = \emptyset \).

Next, we define similarity between grid-based signatures \( S(q) \) and \( S(o) \) as follows. For example, Figure 5 shows the grid-based signature of \( o \) and \( q \).

**Definition 4** (Grid-Based Signature). Given object \( o \), the grid-based signature of \( o \) is the grid in \( G \) intersecting with region \( o.R \), i.e., \( S(o) = \{g \in G \mid g.R \cap o.R \neq \emptyset\} \).

For example, Figure 4 shows the grid-based signature of object \( o_2 \), \( S(o_2) = \{g_6\} \).

Next, we define similarity between grid-based signatures \( S(q) \) and \( S(o) \) as weight summation of their common grids, i.e.,

\[
\operatorname{sim}(S(q), S(o)) = \sum_{g \in S(q) \cap S(o)} w(g | q, o),
\]

where \( w(g | q, o) \) is the weight of grid \( g \) with respect to query \( q \) and object \( o \). Intuitively, the weight captures the degree of spatial similarity between \( q \) and \( o \) contributed by \( g \).

It is not straightforward to define grid weight due to the following reason. Ideally, if grid \( g \) is shared by \( q \) and \( o \), the weight should be the area of the intersecting region of \( q \) and \( o \) in the grid, i.e., \( w(g | q, o) = |g.R \cap q.R \cap o.R| \).

However, in practice, it is expensive to compute \( |g.R \cap q.R \cap o.R| \) at the query time. Thus, we propose to employ an upper bound of \( |g.R \cap q.R \cap o.R| \) to estimate the weight as

\[
w(g | q, o) = \min \{w(g | q), w(g | o)\},
\]

where \( w(g | q) \) (\( w(g | o) \)) is the weight of grid \( g \) with respect to query \( q \) (object \( o \)), and can be estimated by \( w(g | o) = |g.R \cap o.R| \) or \( w(g | q) = |g.R \cap q.R| \).

In addition, we define the threshold \( \tau_k \) to be the area of region \( q.R \) multiplied by threshold \( \tau_k \), i.e., \( \tau_k \cdot |q.R| \). Now, we can prove that the grid-based signature satisfies the key property of signatures, as shown in Lemma 1.

**Lemma 1.** Spatial similarity satisfies \( \operatorname{sim}(q, o) \geq \tau_k \), only if \( \operatorname{sim}(S(q), S(o)) \geq \epsilon_k \).

**Proof.** The proofs are in our technical report. □

For example, in Figure 4, based on the grids \( \{g_1, \ldots, g_{15}\} \), we generate grid-based signatures, \( S(q) \) and \( S(o_2) \), for query \( q \) and object \( o_2 \). We also compute grid weights, e.g., \( w(g_{10} | q) = 750 \), \( w(g_{10} | o_2) = 450 \). Thus, we obtain signature similarity \( \operatorname{sim}(S(q), S(o_2)) = 1375 \). Obviously, we have \( \operatorname{sim}(S(q), S(o_2)) \geq \tau_k \cdot |q.R| = 600 \).
4.2 Threshold-Aware Pruning

Based on grid-based signatures, a straightforward method to filter dissimilar objects is to simply employ the algorithm Sig-Filter in Figure 1. The algorithm probes inverted lists of grids in signature $S(q)$, and inserts the objects satisfying $\text{sim}(S_a(q), S_o(a)) \geq c_o$ into candidate set $C$. However algorithm Sig-Filter has the following limitations. Firstly, for a large query region $q.R$, since it may generate many grids as signature for $q$, the algorithm needs to probe many inverted lists, which may be very expensive. Secondly, when probing the inverted list of a grid, we have to retrieve all objects in the list, resulting in high probing costs for long inverted lists.

To address the above-mentioned limitations, we devise a threshold-aware pruning technique in this section. The objective of our technique is two-fold: 1) to reduce the number of probed inverted lists; and 2) to reduce the number of objects retrieved from a probed inverted list. To this end, we employ the prefix-filtering technique mentioned in Section 3.2.

To use prefix filtering, we first fix a global order on the generated signature elements of the objects in $O$. Then, we sort the elements of every object based on the global order. In the filter step, for each object $o$, instead of using signature $S(o)$, we only select a prefix of the signature, denoted by $S^p(o)$. Similarly, we also select a signature prefix for the query $S^p(q)$. The signature prefixes must satisfy that $\sum_{s \in S^p(q) \cap S^p(o)} w(s) < c$ if $S^p(q) \cap S^p(o) \neq \emptyset$.

Prefix Selection. Given a global order of signature elements, consider signature $S(o) = \{s_1, s_2, \ldots, s_{|S(o)|}\}$ of object $o$, where $s_i$ is the $i$-th element based on the global order. To select prefix $S^p(o) = \{s_1, \ldots, s_p\}$, we can remove the last elements with weight summation smaller than $c$ and select the remaining elements as the prefix, as shown in Lemma 2.

**Lemma 2.** Given a similarity threshold $c$, the signature prefix $S^p(o) = \{s_1, \ldots, s_p\}$ can be selected as

$$p = \min \{i\} \text{ s.t. } \sum_{j=p+1}^{|S(o)|} w(s) < c. \tag{2}$$

Grid Order. In this paper, we sort the grids in ascending order of the number of the object regions intersecting with them. Formally, let $\text{count}(g)$ denote the number of object regions in $O$ intersecting with grid $g$, i.e., $\text{count}(g) = |\{R \cap g \neq \emptyset\}|$. Then, we sort all grids in $G$ in ascending order of $\text{count}(g)$. For example, in Figure 2 we sort the grids intersecting with $q.R$ by $\{g_7, g_{10}, g_{11}, g_{14}, g_{15}, g_9\}$. Observed from this figure, we have $c_6 = 600$, $w(g_{15} \mid q) = 300$ and $w(g_9 \mid q) = 250$. According to Lemma 1, we can select signature prefix as $\{g_7, g_{10}, g_{11}, g_{14}\}$, i.e., $p = 4$, because any shorter prefix ($p < 4$) may cause the reduction of weights larger than threshold $c_6$, i.e., $\sum_{j=p+1}^{|S(o)|} w(g \mid q, o) \geq c_6$.

Therefore, instead of considering all elements in $S(q)$, we only need to probe the inverted lists of the ones in $S^p(q)$, which can reduce the filtering complexity.

Inverted Index with Threshold Bounds. To further improve the performance, we propose to reduce the number of retrieved objects in a probed inverted list. Recall that, for an object $o$, we only need to consider the signature elements in prefix $S^p(o)$ rather than $S(o)$. Thus, when probing the inverted list $I(s)$ of element $s \in S(o)$, we only need to retrieve the objects in $I(s)$ containing $s$ in their signature prefixes given threshold $c$, i.e., $I^c(s) = \{o \mid o \in I(s), s \in S^p(o)\}$.

A challenging problem is to efficiently compute $I^c(s)$ based on various thresholds $c$ for different queries. To address this problem, we augment a threshold bound to each object in every inverted list. Specifically, for an object $o$ in an inverted list $I(s)$ of element $s$, we maintain a threshold upper bound $\tau_s(o)$, which represents the maximum threshold that we keep for $s$ in $o$’s signature prefix. Thus, if threshold $c > \tau_s(o)$, object $o$ can be pruned from $I^c(s)$, as shown below.

**Lemma 3.** Let $s_i$ be the $i$-th signature element in $S(o)$ (where $1 \leq i \leq |S(o)|$) and $c$ be a signature similarity threshold. The object $o$ can be pruned from $I^c(s_i)$ if

$$c > \tau_s(o) = \sum_{j=1}^{|S(o)|} w(s_j). \tag{3}$$

We store bound $\tau_s(o)$ for each object $o$ in inverted list $I(s)$, and sort the objects in descending order of the bounds. Thus, given a threshold $c$, we can efficiently find $I^c(s) = \{o \mid o \in I(s), \tau_s(o) \geq c\}$, when probing the inverted list of element $s$. Figure 2 provides the inverted lists of eight grids. For example, the inverted list of grid $g_{14}$ contains objects $\{o_1, o_2\}$, each of which is associated with a threshold bound, e.g., $\tau_{g_{14}}(o_1) = 900$. Given threshold $c_6 = 600$, we only retrieve $o_1$ when probing the inverted list of $g_{14}$, because bound $\tau_{g_{14}}(o_2) = 550 < c_6$.

Threshold-Aware Pruning. Based on the technique mentioned above, we devise an improved filtering algorithm Sig-Filter+ in Figure 1. Compared with algorithm Sig-Filter, Sig-Filter+ only selects signature prefix $S^p(q)$ for query $q$. For each element $s \in S^p(q)$, it only retrieves objects in $I^c(s)$ instead of $I(s)$, and merges the objects to the candidates $C$. We use the following example to illustrate how algorithm Sig-Filter+ works using grid-based signatures.

**Example 1.** Consider the objects $O$ and query $q$ with thresholds $\tau_q = 0.25$ and $\tau_q = 0.3$ in Figure 5. We generate grid-based signatures and build the inverted index with threshold bounds as shown in Figure 4. Given query $q$, we first generate its signature prefix $S^p_q(q) = \{g_7, g_{10}, g_{11}, g_{14}\}$ based on threshold $c_6 = 600$ according to Lemma 1. Then, for each element $s_k(q)$, we probe its inverted list and only retrieve the objects with bounds no smaller than $c_k$, and obtain the candidates $C_q = \{o_1, o_2, o_3, o_4\}$. Finally, algorithm Sig-Verify reports the answer of $q$, i.e., $A = \{o_2\}$.

---

Function $\text{Sig-Filter}^+(q, I)$

**Input:** $q$: A query; $I$: An inverted index

**Output:** $C$: Candidate objects

1. **begin**
2. Initialize candidate set $C \leftarrow \emptyset$;
3. Generate signature, $S(q) \leftarrow \text{GenSig}(q)$;
4. Compute signature similarity threshold $c$;
5. Select prefix $S^p(q)$ for query $q$;
6. For each element $s \in S^p(q)$ do
7. Find the objects $I^c(s)$ from inverted list $I(s)$;
8. $C \leftarrow C \cup I^c(s)$;
9. **end**

**Figure 6:** Filtering with threshold-aware pruning

---

Note that the global grid order will influence the performance of filtering. We do not study the problem in this paper due to the space limitation, and take it as future work.
Algorithm Sig-Filter\textsuperscript{+} can be also applied to textual signatures. Specifically, we can sort tokens in descending order of their ids, and build the inverted index with threshold bounds. Given query $q$, we only consider the tokens in the signature prefix, denoted by $S_0(q)$, and only retrieve the objects with thresholds no smaller than $c_0$ in each inverted list. For example, we only retrieve inverted lists of $t_1$ and $t_3$ in Figure 3, and obtain candidates $C_T = \{a_1, a_2, a_3, a_4, a_5\}$. Notice that algorithm Sig-Filter\textsuperscript{+} may produce different sizes of candidates when using different signatures.

4.3 Grid Granularity Selection

An essential task in algorithm Sig-Filter\textsuperscript{+} is to generate grid-based signatures, i.e., GenSig. Obviously, the performance of Sig-Filter\textsuperscript{+} is affected by grid granularity and a key challenge is to select an appropriate grid granularity. More specifically, coarse granularity achieves high filtering performance but weakens the filtering power and leads to low verification performance. On the contrary, the fine granularity reduces the number of candidates, but leads to low filtering performance. For instance, in Example 3, $o_5$ is a candidate of query $q$ as $o_5.R$ and $q,R$ share grid $g_{15}$. However, the two regions do not intersect with each other at all, and thus $o_5$ can be actually pruned.

To alleviate the problem, we propose a method for selecting grid granularity in this section. We introduce a probabilistic model to measure the expected query cost for grids of specific granularity. To answer a query $q$, the overall cost $cost(q)$ consists of filtering cost $cost_F(q)$ and verification cost $cost_V(q)$, i.e., $cost(q) = cost_F(q) + cost_V(q)$. The filtering cost $cost_F(q)$ depends on the number of objects retrieved from the inverted index, i.e., $cost_F(q) = \pi_1 \cdot \sum_{g \in S_k(q)} |I'(g)|$, where $\pi_1$ is the average cost of retrieving an object from an inverted list and merging it to candidates. On the other hand, verification cost $cost_V(q)$ depends on the number of candidates, i.e., $cost_V(q) = \pi_2 \cdot |C|$, where $\pi_2$ is the average cost of verifying an object. Thus, we have $cost(q) = \pi_1 \cdot \sum_{g \in S_k(q)} |I'(g)| + \pi_2 \cdot |C|$. For example, we have $cost(q) = 6\pi_1 + 4\pi_2$ for query $q$ in Figure 3.

Notice that the above analysis is based on a single query. To analyze the expected query cost of grid set $G$ with specific granularity, we suppose that we have a query workload $Q$. Then, each grid $g \in G$ has a probability $P(g)$ representing the likelihood that $g$ is used by queries. In addition, when inverted list $I(g)$ is probed by different queries, the returned objects $I'(g)$ may be different. For ease of analysis, we consider the worst case that all objects need to be returned, i.e., $|I'(g)| = |I(g)|$. Thus, we can estimate the expected query cost of all grids in $G$ with the specific granularity as

$$
\text{cost}(G) = \pi_1 \cdot \sum_{g \in G} P(g) \cdot |I(g)| + \pi_2 \cdot |C|, \quad (4)
$$

where $|C|$ is the average size of candidates given the query workload. Now, we can define the grid granularity selection problem: Find the best set of grids $G$ with specific granularity that minimize the expected cost $\text{cost}(G)$.

Since it is intractable to solve the problem by considering arbitrary grid partition schemes, we devise an approximate algorithm for selecting grid granularity as illustrated in Figure 4. The basic idea is to decompose the underlying space $\mathcal{R}$ into a grid tree with height $H$, where the grids in level $l$, denoted by $G^l$, is obtained by a $2^l \times 2^l$ partition of space $\mathcal{R}$. For example, at level 0, there is 1*1 grid, level 1 partitions $\mathcal{R}$ into 2*2 grids, and so forth. Therefore, we reduce grid granularity selection to the problem of finding the best level $l'$ in the grid tree to minimize the expected cost.

We devise an approximate algorithm to solve this problem. We traverse the grid tree from the root to its leaves, and compute the expected cost of each level. For two adjacent levels, $l$ and $l+1$ ($0 \leq l < H$), we compute the benefit of the partitioning as $B(l,l+1) = \text{cost}(G^l) - \text{cost}(G^{l+1})$. If the benefit is smaller than a threshold $\mathcal{B}$, the algorithm terminates, where $\mathcal{B} > 0$ is used to balance efficiency and storage. Then, we prove that for any $\mathcal{B}$, we can find a level $l$, which satisfies that for any level $l > l'$ the filtering benefit $B(l,l+1) < \mathcal{B}$ as formalized in Lemma 4.

**Lemma 4.** $\sqrt{\mathcal{B}} > 0$, there exists a level $l'$ which satisfies that for levels $l > l'$ filtering benefit $B(l,l+1) < \mathcal{B}$.

We briefly show the correctness of Lemma 4 (The proof is in [S]). Based on Equation (3), we have $B_P = \sum_{g' \in G^l} B_F(g')$, where $B_F(g')$ is the benefit of partitioning grid $g'$ into fine-grained grids. We use the example of partitioning $g_1$ into $\{g_1^1, g_1^2, g_1^3, g_1^4\}$ to show how to compute $B_F(g')$. Based on the probabilistic theory, we have $P(g_1^1) = \sum_i P(g_1^i) = \sum_{i \neq j} P(g_1^j) + \sum_{i,j,k} P(g_1^i g_1^j g_1^k) - P(g_1^i g_1^j g_1^k g_1^l)$. Since no query region can intersects with three grids, $P(g_1^i g_1^j g_1^k) = 0$. We denote $\sum_{i \neq j} P(g_1^i g_1^j) + P(g_1^j g_1^k g_1^l)$ as $\theta$. Thus, the benefit of partitioning $g_1^i = \pi_1 \cdot (\sum_{j \neq i} P(g_1^j) \cdot (|I(g_1^i)| - |I(g_1^j)|) - \theta \cdot |I(g_1^i)|)$. With the increase of $l$, the benefit of partitioning a grid becomes less and less significant, since 1) $|I(g_1^i)| = |I(g_1^{i+1})|$ eventually decreases, and 2) $\theta$ eventually increases since it becomes more likely that queries intersect with more grids. Thus, we can find a level $l$ which satisfies that for levels $l > l'$ filtering benefit $B_P(l,l+1) < \mathcal{B}$.

Verification benefit $B_V$ also has the similar property. However, it is difficult to analyze $B_V$, as estimating the average candidate size $|C|$ is very hard. Thus, we take theoretical analysis as a future work and only show the experimental results in Section 1.

5. Hybrid Filtering Algorithms

In this section, we develop hybrid filtering algorithms to simultaneously utilize textual and spatial signatures. We first introduce the hash-based hybrid signature and present a filtering algorithm based on the signature in Section 5.1, and then propose the hierarchical hybrid signature to further improve the performance in Section 5.2.

5.1 Hash-Based Hybrid Signature

A straightforward method is to respectively apply algorithm Sig-Filter\textsuperscript{+} using textual and grid-based signatures, and compute the intersection of candidate sets $C_T$ and $C_R$. 

![Figure 7: Grid Granularity Selection](image-url)
This method, although reducing the number of candidates, may increase the number of objects retrieved from inverted indexes, and thus result in higher filtering cost.

In order to reduce the filtering cost, we propose the hash-based hybrid signature to efficiently find possibly similar objects. The basic idea is to hash tokens and grids of each same object into buckets using a hashing function, and then use each bucket as a hybrid signature element.

Definition 5 (The Hash-Based Hybrid Signature). For each object \( o \) with textual signature \( S_t(o) \) and grid-based signature \( S_g(o) \), the hash-based hybrid signature of \( o \) is defined by \( S_h(o) = \{ h = (t, g) \mid t \in S_t(o), g \in S_g(o) \} \), where \( h \) is a hash value by hashing \( t \) and \( g \) into a bucket.

Based on hash-based hybrid signatures, we develop a more efficient filtering algorithm Hybrid-Sig-Filter\(^+\) in Figure 9. The algorithm respectively generates textual and grid-based signatures \( S_t(o) \) and \( S_g(o) \) for each object \( o \in \mathcal{O} \). It hashes tokens in \( S_t(o) \) and grids in \( S_g(o) \) into buckets to generate a hybrid signature \( S_h(o) \). Then, the algorithm builds an inverted list \( \mathcal{I} \) for the hybrid signatures generated from all objects in \( \mathcal{O} \). Compared with the inverted list introduced in Section 4.3, we augment both spatial and textual threshold bounds for each object \( o \) in each inverted list of element \( h \), denoted by \( \tau^S_h(o) \) and \( \tau^T_h(o) \). The two bounds can be computed according to Lemma 5, and satisfy that if either \( \tau^S_h(o) > \tau^T_h(o) \) or \( \tau^T_h(o) > \tau^S_h(o) \), \( o \) can be safely pruned from the inverted list of element \( h \). Moreover, to avoid generating too many inverted lists, we introduce a constraint of index sizes to guarantee that the number of hash buckets is smaller than a given number, which is explained in Section 5.2.

Given query \( q \), the algorithm respectively generates textual and grid-based signatures \( S_t(q) \) and \( S_g(q) \), and computes signature similarity thresholds \( c_t \) and \( c_g \). Then, it respectively selects prefixes \( S^T_h(q) \) and \( S^S_h(q) \). Next, for each token \( t \in S^T_h(q) \), the algorithm examines each grid \( g \in S^S_h(q) \), and computes the hash-based signature element \( h = (t, g) \). Using element \( h \), the algorithm probes its inverted list \( \mathcal{I}(h) \) and retrieves the objects satisfying \( \tau^S_h(o) \geq c_t \) and \( \tau^T_h(o) \geq c_g \), denoted by \( \mathcal{I}^{(c_t, c_g)}(h) \). Finally, the algorithm merges the retrieved objects \( \mathcal{I}^{(c_t, c_g)}(h) \) to the candidate set \( C \).

Example 4. Figure 9 shows how algorithm Hybrid-Sig-Filter\(^+\) works. For each object, the algorithm hashes its tokens and grids into buckets to generate a hybrid signature. For example, the signature of object \( o_1 \) consists of elements such as \((t_1, g_{10}), (t_1, g_{11}), (t_1, g_{14})\), etc. Given query \( q \) with prefixes \( S^T_h(q) = \{t_1, t_3\} \) and \( S^S_h(q) = \{g_7, g_{10}, g_{11}, g_{14}\} \), the algorithm obtains hash-based hybrid signatures, and probes the corresponding inverted lists. When probing a list, the algorithm only retrieves the objects satisfying \( \tau^S_h(o) \geq c_t \) and \( \tau^T_h(o) \geq c_g \). For example, the inverted list of element \((t_1, g_{14})\) only returns \( o_1 \). Finally, the algorithm produces candidates \( C = \{o_1, o_2, o_3\} \).

5.2 Hierarchical Hybrid Signatures

Algorithm Hybrid-Sig-Filter\(^+\) generates hybrid signatures using grids with fixed granularity. As different regions can use grids with different granularities, this method has the following limitations. Generating coarse-grained grids for small regions may weaken the filtering power and introduce more candidates. For example, the element \((t_1, g_{11})\) in Figure 9 involves a dissimilar object \( o_2 \) as the estimated grid weight \( w(g \mid o_2) = 200 \) is much larger than the real weight \( w(g \mid o_2, q) = 0 \). On the other hand, generating fine-grained grids for large regions may involve too many useless signature elements, leading to high costs of both storage of inverted lists and filtering. In Figure 9, any fine-grained grid covered by \( g_{14} \) is useless to region \( R_1 \), because \( g_{14} \) has already provided an accurate grid weight for \( R_1 \).

In order to address the problem, we propose to judiciously select hierarchical grids for each token \( t \) given a constraint of index sizes (i.e., the maximum number of hybrid signature elements), and generate hierarchical hybrid signatures to improve the performance. We first formalize the hierarchical hybrid signature selection problem as follows.

Hierarchical hybrid signature selection. Intuitively, our objective is to select at most \( m_t \) hierarchical grids for the objects containing each token \( t \) and optimize the filtering power. As mentioned in Section 4.3, optimizing the filtering power can be reduced to minimizing filter and verification costs. Since verification is the bottleneck as shown in Section 4.2, we focus on minimizing verification cost, i.e., the average size of candidates \( |C| \) in this section. It is known that the estimation of \( |C| \) is very difficult, so we consider a simplified version of the problem.

Ideally, suppose that we have a set of grids with finest granularity, such that each finest grid is totally covered by or exclusive from object regions. Using the finest grids, we can obtain the most compact candidate set satisfying \( |C| = \{|A| \} \). Obviously, the amount of the finest grids must be very huge. Therefore, given a number \( m_t \), we need to merge the finest grids to at most \( m_t \) hierarchical grids. Different

---

**Function Hybrid-Sig-Filter\(^+\) (q, I)**

**Input:** q: A query; I: A hybrid inverted index

**Output:** C: Candidate objects

1. begin
2. Initialize candidate set \( C \leftarrow \emptyset \);
3. Generate signatures \( S_t(q) \) and \( S_g(q) \);
4. Compute signature similarity thresholds \( c_t \) and \( c_g \);
5. Select prefixes \( S^T_h(q) \) and \( S^S_h(q) \);
6. for each token \( t \) in \( S^T_h(q) \) do
7. for each grid \( g \) in \( S^S_h(q) \) do
8. Compute the hybrid signature \( h = (t, g) \);
9. Find object list \( \mathcal{I}^{(c_t, c_g)}(h) \) from \( I \);
10. \( C \leftarrow C \cup \mathcal{I}^{(c_t, c_g)}(h) \);
11. end

**Figure 8:** Hybrid signature based filtering algorithm

<table>
<thead>
<tr>
<th>( S_t(q) )</th>
<th>( S_g(q) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ t1, t3 }</td>
<td>{ g7, g10, g11, g14 }</td>
</tr>
<tr>
<td>\tau^S_h(o) \geq 0.57</td>
<td>\tau^T_h(o) \geq 600</td>
</tr>
</tbody>
</table>

**Figure 9:** Example of algorithm Hybrid-Sig-Filter\(^+\) for small regions may weaken the filtering power and introduce more candidates. For example, the element \((t_1, g_{11})\) in Figure 9 involves a dissimilar object \( o_2 \) as the estimated grid weight \( w(g \mid o_2) = 200 \) is much larger than the real weight \( w(g \mid o_2, q) = 0 \). On the other hand, generating fine-grained grids for large regions may involve too many useless signature elements, leading to high costs of both storage of inverted lists and filtering. In Figure 9, any fine-grained grid covered by \( g_{14} \) is useless to region \( R_1 \), because \( g_{14} \) has already provided an accurate grid weight for \( R_1 \).

In order to address the problem, we propose to judiciously select hierarchical grids for each token \( t \) given a constraint of index sizes (i.e., the maximum number of hybrid signature elements), and generate hierarchical hybrid signatures to improve the performance. We first formalize the hierarchical hybrid signature selection problem as follows.

Hierarchical hybrid signature selection. Intuitively, our objective is to select at most \( m_t \) hierarchical grids for the objects containing each token \( t \) and optimize the filtering power. As mentioned in Section 4.3, optimizing the filtering power can be reduced to minimizing filter and verification costs. Since verification is the bottleneck as shown in Section 4.2, we focus on minimizing verification cost, i.e., the average size of candidates \( |C| \) in this section. It is known that the estimation of \( |C| \) is very difficult, so we consider a simplified version of the problem.

Ideally, suppose that we have a set of grids with finest granularity, such that each finest grid is totally covered by or exclusive from object regions. Using the finest grids, we can obtain the most compact candidate set satisfying \( |C| = \{|A| \} \). Obviously, the amount of the finest grids must be very huge. Therefore, given a number \( m_t \), we need to merge the finest grids to at most \( m_t \) hierarchical grids. Different
the algorithm does not split the node and only inserts
approximately computes its error based on the child nodes,
partitioning problem in [15].
If a node is an intermediate node, the algorithm examines whether to
split it, and enqueues its child nodes of 

\[ n \rightarrow \text{Dequeue}() \]

if \( n \) is a leaf node then 
\[ G_t \cup \{ n \} \]
else for Each child node \( n_c \) in \( N_c \) do
\[ Q.\text{Enqueue} (n_c, \text{Error}(n_c)) \]
13 end

Figure 11: Greedy Algorithm for the HSS problem.

Algorithm 2: HSS-Greedy \((I(t), m_t)\)

Input: \( I(t) \): objects containing token \( t \); \( m_t \): A number
Output: \( G_t \): The selected grids
1 begin
2 Construct a grid tree \( GT \) for objects in \( I(t) \); 
3 Initialize a priority queue \( Q \); 
4 \( Q.\text{Enqueue} (\text{GT.root}, \text{Error}(\text{GT.root})) \); 
5 while \( Q \) is not empty do
6 \[ n \leftarrow Q.\text{Dequeue}() \]
7 if \( n \) is a leaf node then 
\[ G_t \cup \{ n \} \]
8 else
9 \[ N_c \leftarrow \text{Child nodes of } n \]
10 if \( |G_t| + |N_c| - 1 > m_t \) then 
\[ G_t \cup \{ n \} \]
11 else for Each child node \( n_c \) in \( N_c \) do
12 \[ Q.\text{Enqueue} (n_c, \text{Error}(n_c)) \]
13 end

Figure 10: Hierarchical hybrid signatures of \( t_1 \)
grids merged from the finest grids may lead to different up-
per bounds of grid weights (See Section 3.4) and result in
different candidate sizes. To measure quality of the grids, we
introduce the error of each grid. Formally, we define the er-
ror as follows. Consider a grid \( g \) with inverted list \( I(g) \). Us-
ing uniform assumption, we can estimate the expected size of
inverted list as \[ |I(g)| = \sum_{g \in I(q)} \text{In} \frac{|g \cap R|}{|R|} \] is the probability that query region \( q \) intersects with each object
region \( o.R \). For example, consider grid \( g_{13} \) in Figure 1. We
can compute \[ |I(g_{13})| = (|g_{13} \cap R_1| + |g_{13} \cap R_2|) / |g| = 1.7 \]

Definition 6 (Error of Grid). Consider grid \( g \) covering
finest grids \( g_1, \ldots, g_3 \). The error of \( g \), denoted by
\( \text{Error}(g) \), is \[ \sum_{g_i} (|I(g)| - |I(g_i)|)^2 \].

Based on the errors of grids, we formally define our hierar-
chical hybrid signature selection (HSS) problem as follows.

Definition 7 (The HSS Problem). Given a token \( t \) and a
number \( m_t > 1 \), find at most \( m_t \) hierarchical grids \( G_t \)
such that \[ \sum_{g \in G_t} \text{Error}(g) \] is minimized.

We can prove that the HSS problem is NP-hard by a re-
duction from a known NP-hard problem, a rectangular par-
titioning problem in [15].

Theorem 1. The HSS problem is NP-Hard.

To solve the HSS problem, we propose a heuristic algo-
rithm to find the grids with the minimum errors, as shown in
Figure 11. Given all objects indexed by a token \( t \), i.e., \( I(t) \),
the algorithm first constructs a grid tree, and initializes a
priority queue where elements are sorted in descending or-
dering of their scores. Then, it inserts the root with its grid
error as the score. Here, for a node \( n \), the algorithm
approximately computes its error based on the child nodes,
\[ \text{Error}(n) = \sum_{\text{child}(n)} (|I(n)| - |I(\text{child}(n))|)^2 \]. Next, the
algorithm traverses the grid tree until the queue is empty as
follows. It removes the front element in the queue, denoted by
\( n \). If \( n \) is a leaf node, the algorithm inserts it into \( G_t \). If
\( n \) is an intermediate node, the algorithm examines whether to
split \( n \) using its child nodes \( N_c \). If the number of grids after
the splitting is larger than \( m_t \), i.e., \[ |G_t| + |Q| + |N_c| - 1 > m_t \],
the algorithm does not split the node and only inserts \( n \) into
\( G_t \); otherwise, the algorithm inserts the child nodes of \( n \) into the queue.

Figure 11 provides an example to show how algorithm
HSS-Greedy generates hierarchical hybrid signatures for re-
gions \{\( R_1, R_2, R_3 \)\} of token \( t_1 \). The algorithm first enqueues
the root grid \( g_{13}^1 \), i.e., the space \( R \). Then, the algorithm
repeatedly dequeues the front element, and enqueues its child
nodes. For example, since \[ \text{Error}(g_{13}^1) = 0.1 > \text{Error}(g_{13}^2) = 0.09 \], the algorithm
deqeues grid \( g_{13}^2 \) and enqueues its child nodes. Thus, given \( m_t = 8 \), we obtain
hierarchical hybrid signatures represented as bold circles in Figure 11.

Hierarchical signature-based filtering algorithm. Using
the generated hierarchical hybrid signatures, we can im-
prove the performance of algorithm Hybrid-Sig-Filter\(^+\) (in Figure 8) as follows. For each token \( t \), we first fix a
global order of its hierarchical grids. We sort the grids in
ascending order of their levels in the grid tree. For grids in
the same level, we sort them in ascending order of the num-
ber of object regions intersecting with them. For example,
based on the global order, the hierarchical hybrid signature
of token \( t_1 \) can be sorted as \( \{ g_1^3, g_2^3, g_3^3, g_4^3, g_5^3, g_6^3 \} \). Then,
we can employ algorithm Hybrid-Sig-Filter\(^+\) to filter dis-
similar objects, as illustrated in the following example.

Example 5. Figure 11 provides an example of algorithm
Hybrid-Sig-Filter\(^+\) using hierarchical hybrid signatures.
Consider token \( t_1 \) in textual signature prefix \( S_T^1(q) \). We
can generate three hierarchical grids \( \{g_1^3, g_2^3, g_3^3\} \) and compute the
weight \( w(g \mid q) \) for each grid. Using threshold-aware prun-
ing, we can prune inverted lists of grids \( g_1^3 \) and \( g_2^3 \), and only
need to probe the inverted list of hybrid signature \( (t_1, g_3^3) \).
Similarly, we probe the inverted lists of other tokens in \( S_T^1(q) \)
and obtain a candidate set \( C = \{ a_1, a_2 \} \). Notice that the
candidate set is more compact than the one in Example 4.

6. EXPERIMENTS

In this section, we report experimental results. We ex-
tended the state-of-the-art spatial keyword search method
IR-tree [4] to support spatio-textual similarity search as
mentioned in Section 2.3. We compared our SEAL method
with this method.

6.1 Experiment Setup

We used two datasets. The first one was a real dataset
Twitter. We collected 60 million tweets from May 2011 to
August 2011, among which about 13 million tweets had
locations (i.e., points with longitudes and latitudes). We ran-
domly selected 1 million users, and obtained ROI objects
from their profiles. On the one hand, we selected frequent words in her tweets as the token set for each user, and the average token number of objects was 14.3. On the other hand, we took the MBR of all of her tweets as each user’s active region, and the average area of the regions was 115 square kilometers (sq.km.). Specifically, the distributions of the regions were: 0.0001 sq.km. (4.4%), 0.01 sq.km. (15.4%), 1 sq.km. (29.7%), 100 sq.km. (73%), etc, where “x sq.km. (y%)” means there are y% regions with areas not greater than x sq.km. We can see that the regions had various sizes due to different spatial distributions of users’ tweets, and most of regions sizes were not very large. Note that more complicated methods can be used to obtain the active regions from users’ tweets. For example, we can compute multiple active regions for each user by clustering tweets’ locations. We take it as a future work, and only focus on the general spatio-textual similarity search problem in this paper. In addition, we generated two query sets to evaluate the performance of different algorithms. Each query set contained 100 queries.

**Large-Region Queries:** We generated a set of queries with large regions. The average query token number was 6.97, and the average area of regions was 554 sq.km, which was equivalent to the area of a district.

**Small-Region Queries:** We generated a set of queries with small regions. The average query token number was 12.9, and the average area of query regions was 0.44 sq.km, which was equivalent to the area of a small neighborhood.

We also used a synthetic dataset by combing POIs in USA and the publication records in DBLP. We selected 1 million POIs from the USA dataset as centers and extended the POIs with random widths and heights to generate regions. Then, we randomly distributed publication records to the regions to generate token sets. The average area of regions was 0.4 sq.km and the average token number was 12.5. We also generated 100 large-region queries and 100 small-region queries for this dataset. Note that the experimental results on the two datasets were similar. Due to the space constraints, we only provide experimental results of method comparison on the USA data set in Section 6.5.

The IR-tree index was disk-resident, and its page size was 4KB. The inverted indexes of our signature-based methods were also disk-resident, and we maintained an index that mapped each signature element to the disk offset of its inverted list in memory. Note that this index was small enough to be maintained in memory. For example, for the Twitter dataset, the index only occupied 19 MB. Table 1 summarizes the data statistics and index sizes. For simplicity, we respectively use TokenInv, GridInv, HashInv and HierarchicalInv to represent inverted indexes of textual, grid-based, hash-based hybrid and hierarchical hybrid signatures. For GridInv and HashInv we also present the granularity. For example, GridInv (1024) represents the GridInv index with granularity 1024 × 1024. Due to space constraints, we only show sizes of the indexes we used in method comparison in Section 6.5. Besides, we varied both spatial and textual similarity thresholds from 0.1 to 0.5, and the default value of the thresholds was 0.4.

In the paper, we only show the running time and the numbers of candidate numbers of different methods are in our technical report [6].

All the programs were implemented in JAVA and all the experiments were run on the Ubuntu machine with an Intel Core 2 Quad X5450 3.00GHz processor and 4 GB memory.

### 6.2 Token Filter vs. Grid Filter

We first evaluated algorithm Sig-Filter⁺ in Figure 8 using textual signatures and grid-based signatures, which are respectively denoted by TokenFilter and GridFilter. We examined the GridFilter using grid-based signatures with different granularities. Figure 12 provides the experimental results. We can see that the performance of TokenFilter and GridFilter depended on the similarity thresholds, $\tau_x$ and $\tau_\text{IR}$. Observed from Figure 12(a), TokenFilter outperformed GridFilter for small $\tau_\text{IR}$. However, with the increase of $\tau_\text{IR}$, GridFilter became faster. For example, given $\tau_x = 0.5$, GridFilter (1024) took 30 milliseconds, while TokenFilter took 64 millisecond. Similarly, with the increase of $\tau_x$, TokenFilter became better (See Figures 12(b) and 12(d)). Therefore, it is better to combine both filters instead of using either one individually.

### 6.3 Evaluating Different Grid Granularities

We then evaluated the performance of GridFilter with different grid granularities. We partitioned the entire space into $p \times p$ uniform grids as mentioned in Section 6.2, where $p$ denotes the granularity. Then, we ran GridFilter based on different granularities and compared the filtering time and verification time. Figure 13 shows the experimental results. We can see that with the increase of granularity, the verification time always decreased. For example, the verification time decreased from 466 milliseconds at granularity 64 to 90 milliseconds at granularity 8192 in Figure 13(b). Moreover,
the decrease of verification cost became more and more insignificant as the granularity increased. The filtering step was not always improved with the increase of granularity, which was consistent with our cost-based analysis in Section 6.4. For example, in Figure 13(a), the filtering time first decreased and then increased after granularity 1024.

Figure 13: Evaluation on grid granularity selection on the Twitter data set.

6.4 Evaluating Hybrid Filtering Algorithms

We evaluated hybrid filtering algorithms mentioned in Section 6.4. We first compared the algorithm using hash-based hybrid signatures (HybridFilter) with GridFilter. Figure 14 shows the experimental results, where G and H respectively stand for GridFilter and HybridFilter. We can see that HybridFilters significantly outperformed GridFilters for both large-region and small-region queries. For example, in Figure 14(a), HybridFilters with different granularities (i.e., 256, 512, 1024) were one order of magnitude faster than GridFilters with the same granularity. This is because HybridFilter utilized both spatial and textual pruning simultaneously and had larger filtering power than GridFilter with only spatial pruning.

Figure 14: Comparison of hybrid signatures on the Twitter data set ($\tau = 0.4, \tau_t = 0.1$).

Then, we evaluated the effect of hierarchical hybrid signatures, which were judiciously selected to improve the performance of hybrid filtering. We compared filtering algorithms with hash-based hybrid signatures and hierarchical hybrid signatures given different constraints of index sizes (i.e., maximum numbers of signature elements as mentioned in Section 6.4). Figure 15 shows the experimental results. Hierarchical hybrid signatures achieved better performance compared with hash-based hybrid signatures in various index sizes. For example, in Figure 15(b), the elapsed time of algorithm with hierarchical hybrid signatures was 30 milliseconds given index size 280 MB, while that of hash-based hybrid signatures was 70 milliseconds. The main reason is that we judiciously selected hybrid signatures with hierarchical grids. These signatures improved the filtering power, and thus pruned a large number of dissimilar objects.

6.5 Comparison with Existing Methods

We compared our algorithm using hierarchical hybrid signatures (denoted by SEAL) with the keyword-first method (Keyword), the spatial-first method (Spatial), and state-of-the-art spatial keyword search method IR-tree as discussed in Section 6.6. Figures 16 and 17 respectively show the experimental results on Twitter and USA datasets.

We can see that Keyword and Spatial could not effectively prune dissimilar objects. Specifically, Keyword sometimes performed worse than Spatial (See Figure 6(d)), as it did not have spatial pruning power. On the other hand, for large textual thresholds, Spatial might achieve lower efficiency (See Figures 6(d) and 7(d)), as it did not have textual pruning power. In addition, IR-tree also achieved low performance, and it was even worse than Spatial (See Figures 10). This is because the method, which is designed for spatial keyword search, visited too many unnecessary nodes and involved a huge number of dissimilar objects as mentioned in Section 6.4. In addition, IR-tree had to probe the inverted file associated in each R-tree node, resulting in a large overhead.

Our method SEAL always achieved the highest performance for any type of queries and any threshold on the two datasets. In Figures 16 and 17, our method was several tens of times faster than the baseline methods. For example, in Figure 16(c) given spatial threshold 0.1 and textual threshold 0.4, our method took 5 milliseconds, while IR-tree, Keyword and Spatial respectively took 253, 52 and 182 milliseconds. The better performance of our method is attributed to the signature-based methods integrating both spatial and textual pruning simultaneously, and the hierarchical hybrid signatures we selected for large filtering power.

6.6 Scalability

We evaluated the scalability of our hybrid filtering algorithm by varying the numbers of objects. Figure 18 shows the results. We can see that our method scaled very well, and with the increase of the numbers of objects, the elapsed time increased sub-linearly. This is because our algorithm could prune a huge amount of dissimilar objects even if the number of objects increases.

Figure 18: Scalability on the Twitter data set.

7. CONCLUSION AND FUTURE WORK

In this paper, we have studied a new research problem called spatio-textual similarity search. We introduced a filter-and-verification framework to compute the answers. We devised efficient signature-based filtering algorithms and developed effective pruning techniques. For spatial pruning, we proposed grid-based signatures by decomposing the underlying space, and developed threshold-aware pruning techniques. To utilize spatial and textual pruning simultaneously, we judiciously selected hierarchical hybrid signatures and devised hybrid filtering algorithms. We have implemented our method and examined it on real and synthetic datasets. Experimental results show that our method achieves very high search performance.
REFERENCES