Efficient Processing of Top-k Spatial Keyword Queries

João B. Rocha-Junior*, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg

Department of Computer and Information Science
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
{joao, orestis, simonj, noervaag}@idi.ntnu.no

Abstract. Given a spatial location and a set of keywords, a top-k spatial keyword query returns the k best spatio-textual objects ranked according to their proximity to the query location and relevance to the query keywords. There are many applications handling huge amounts of geotagged data, such as Twitter and Flickr, that can benefit from this query. Unfortunately, the state-of-the-art approaches require non-negligible processing cost that incurs in long response time. In this paper, we propose a novel index to improve the performance of top-k spatial keyword queries named *Spatial Inverted Index* (S2I). Our index maps each distinct term to a set of objects containing the term. The objects are stored differently according to the document frequency of the term and can be retrieved efficiently in decreasing order of keyword relevance and spatial proximity. Moreover, we present algorithms that exploit S2I to process top-k spatial keyword queries efficiently. Finally, we show through extensive experiments that our approach outperforms the state-of-the-art approaches in terms of update and query cost.

1 Introduction

Given a location and a set of keywords, a top-k spatial keyword query returns a ranked set of the k best spatio-textual objects taking into account both 1) the spatial distance between the *objects* (spatio-textual objects) and the query location, and 2) the relevance of the text describing the objects to the query keywords. There are several applications that can benefit from top-k spatial keyword queries such as finding the tweets sent from a given location (Twitter) or finding images near by a given location whose annotation is similar to the query keywords (Flickr). There are also other applications for GPS-enabled mobile phones that can benefit from such queries.

For example, Fig. 1 shows a spatial area containing objects p (bars and pubs) with their respective textual description. Consider a tourist in São Paulo with a GPS mobile phone that wants to find a bar playing *samba* near her current location q. The tourist poses a top-3 spatial keyword query on her mobile phone

* On leave from the State University of Feira de Santana (UEFS).

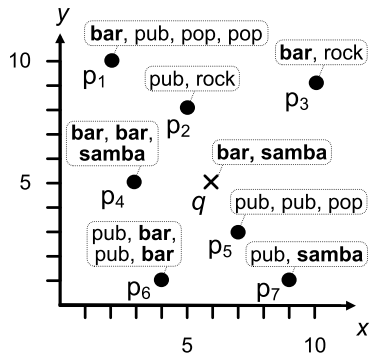


Fig. 1. Example of top-k spatial keyword query.

with the keywords *bar* and *samba* (the query location q is automatically sent by the mobile phone). The top-1 result is p_4 because its description is similar to the query keywords, and it is close to the query location q. The top-2 result is p_6 that is nearer to q than p_7 and has a better textual relevance to the query keywords than p_7. Here, we are assuming for simplicity that documents with higher numbers of occurrences of query keywords are more textually relevant. Later, we will drop this assumption and present a more advanced model. Consequently, the top-3 results are $\langle p_4, p_6, p_7 \rangle$.

Top-k spatial keyword queries are intuitive and constitute a useful tool for many applications. However, processing top-k spatial keyword queries efficiently is complex and requires a hybrid index combining information retrieval and spatial indexes. The state-of-the-art approaches proposed by Cong *et al.* [4] and Li *et al.* [11] employ a hybrid index that augments the nodes of an R-tree with inverted indexes. The inverted index at each node refers to a pseudo-document that represents all the objects under the node. Therefore, in order to verify if a node is relevant for a set of query keywords, the current approaches access the inverted index at each node to evaluate the similarity between the query keywords and the pseudo-document associated with the node. This process incurs in non-negligible processing cost that results in long response time.

In this paper, we propose a novel method for processing top-k spatial keyword queries more efficiently. Instead of employing a single R-tree embedded with inverted indexes, we propose a new index named *Spatial Inverted Index* (S2I) that maps each keyword (term) to a distinct *aggregated R-tree* (*aR-tree*) [14] that stores the objects with the given term. In fact, we employ an aR-tree only when the number of objects exceeds a given threshold. As long as the threshold is not exceeded, the objects are stored in a file, one block per term. However, for ease of presentation, let us assume that we employ an aR-tree for each term. The aR-tree stores the latitude and longitude of the objects, and maintains an aggregated value that represents the maximum term impact (normalized weight) of the objects under the node. Consequently, it is possible to retrieve the best objects ranked in terms of both spatial relevance and keyword relevance effi-

ciently and incrementally. For processing a top-k spatial keyword query with a single keyword, only few nodes of a single aR-tree are accessed. For queries with more than one keyword, we employ an efficient algorithm that aggregates the partial-scores on keyword relevance of the objects to obtain the k best results efficiently. In summary, the main contributions of this paper are:

- We present S2I, an index that maps each term in the vocabulary into a distinct aR-tree or block that stores all objects with the given term.
- We propose efficient algorithms that exploit the S2I in order to process top-k spatial keyword queries efficiently.
- Finally, we show through an extensive experimental evaluation that our approach outperforms the state-of-the-art algorithms in terms of update time, I/O cost, and response time.

The rest of this paper is organized as follows. Sect. 2 gives an overview of the related work. Sect. 3 poses the problem statement. In Sect. 4, we describe S2I. In Sect. 5 and 6, we present the algorithms for processing top-k spatial keyword queries. Finally, the experimental evaluation is presented in Sect. 7 and the paper is conclude in Sect. 8.

2 Related Work

Initially, the research on spatial keyword queries focused on improving the performance of spatial queries in search engines. Zhou *et al.* [17] did a relevant work combining inverted indexes [18] and R-trees [2], and propose three approaches: 1) indexing the data in both R-trees and inverted indexes, 2) creating an R-tree for each term, and 3) integrating keywords in the intermediary nodes of an R-tree. They found out that the second approach achieved better performance. However, they did not consider objects with a precise location (latitude and longitude), and did not provide support for top-k spatial keyword queries. Chen *et al.* [3] also had an information retrieval perspective on their work and did not provide support for exact query location of objects. In their approach, inverted indexes and R-trees are accessed separately in two different stages.

With the popularization of GPS-enabled devices, the research focused on searching for objects in a specific location. Hariharan *et al.* [7] proposed augmenting the nodes of an R-tree with keywords extracted from the objects in the sub-tree of the node. These keywords are then indexed in a structure similar to an inverted index for fast retrieval. Their approach supports conjunctive query in a given region of space. It is not clear, however, how their solution can be extended to support top-k spatial keyword queries. Later, Ian de Felipe *et al.* [5] proposed a data structure that integrates signature files and R-trees. The main idea was indexing the spatial objects in an R-tree employing a signature on the nodes to indicate the presence of a given keyword in the sub-tree of the node. Consequently, at query processing time, the nodes that cannot contribute with the query keywords can be pruned. The main problem of this approach is the limitation to Boolean queries and to a small number of keywords per document.

To the best of our knowledge, there are two previous approaches that support top-k spatial keyword queries. They were developed concurrently by Cong *et al.* [4] and Li *et al.* [11]. Both approaches augment the nodes of an R-tree with a document vector that represents all documents in the sub-tree of the node. For all terms present in the objects in the sub-tree of the node, the vector stores the maximum impact of the term (normalized weight). Consequently, the vector allows computing an upper bound for the textual score (textual relevance) that can be achieved visiting a given node. Hence, it is possible to rank the nodes according to textual relevance and spatial relevance, and decide which nodes should be accessed first to compute the top-k results.

The work of Cong *et al.* goes beyond the work of Li *et al.* by incorporating document similarity to build a more advanced R-tree namely DIR-tree. DIR-tree groups, in the same node, objects that are near each other in terms of spatial distance, and whose textual description are also similar. Furthermore, instead of comparing vectors at query time, DIR-tree employs an inverted index associated with each node that permits to retrieve the children of the node that can contribute with a given query keyword efficiently. Only the posting lists associated with the query keywords are accessed. Cong *et al.* also propose clustering the nodes of DIR-tree (CDIR-tree) to further improve the query processing performance. The main idea is grouping related entries (objects, in case of leaf-nodes) and employing a pseudo-document to represent each group. Hence, more precise bounds can be estimated at query time, consequently, improving the query processing performance. However, it is not clear if the improvement achieved at query processing time compensates the additional cost required for clustering the nodes (pre-processing), and the extra storage space demanded by CDIR-tree. Moreover, keeping a CDIR-tree updated is more complex. For this reason, we decided to compare our approach against the DIR-tree proposed by Cong *et al.*, and we consider this approach as the state-of-the-art.

3 Problem Statement

Let P be a dataset with $|P|$ spatio-textual objects $p = \langle p.id, p.l, p.d \rangle$, where $p.id$ is the identification of p, $p.l$ is the spatial location (latitude and longitude) of p, and $p.d$ is the textual document describing p (e.g., menu of a restaurant). Let $q = \langle q.l, q.d, q.k \rangle$ be a top-k spatial keyword query, where $q.l$ is the query location (latitude and longitude), $q.d$ is the set of query keywords, and $q.k$ is the number of expected results. A query q returns $q.k$ spatio-textual objects $\{p_1, p_2, \cdots, p_{q.k}\}$ from P with the highest scores $\tau(p, q)$, $\tau(p_1, q) \geq \tau(p_2, q) \geq \cdots \geq \tau(p_k, q)$. Furthermore, a spatio-textual object p is part of the result set R of q, if and only if exists at least one term $t \in q.d$ that is also in $p.d$ ($p \in R \Leftrightarrow \exists t \in q.d : t \in p.d$). The *score* of p for a given query q is defined in the following equation:

$$\tau(p, q) = \alpha \cdot \delta(p.l, q.l) + (1 - \alpha) \cdot \theta(p.d, q.d) \tag{1}$$

where $\delta(p.l, q.l)$ is the spatial proximity between the query location $q.l$ and the object location $p.l$, and $\theta(q.d, p.d)$ is the textual relevance of $p.d$ according to

$q.d$. Both measures return values within the range $[0, 1]$. The *query preference parameter* $\alpha \in (0, 1)$ defines the importance of one measure over the other. For example, $\alpha = 0.5$ means that spatial proximity and textual relevance are equally important. In the following, we define the measures more precisely.

Spatial proximity (δ). The spatial proximity is defined in the following equation:

$$\delta(p.l, q.l) = 1 - \frac{d(p.l, q.l)}{d_{max}} \tag{2}$$

where $d(p.l, q.l)$ is the Euclidean distance between $p.l$ and $q.l$, and d_{max} is the largest Euclidean distance that any two points in the space may have. The maximum distance may be obtained, for example, by getting the largest diagonal of the Euclidean space of the application.

Textual relevance (θ). There are several similarity measures that can be used to evaluate the textual relevance between the query keywords $q.d$ and the text document $p.d$ [13]. In this paper, we adopt the well-known *cosine* similarity between the vectors composed by the weights of the terms in $q.d$ and $p.d$:

$$\theta(p.d, q.d) = \frac{\sum_{t \in q.d} w_{t,p.d} \cdot w_{t,q.d}}{\sqrt{\sum_{t \in p.d}(w_{t,p.d})^2 \cdot \sum_{t \in q.d}(w_{t,q.d})^2}} \tag{3}$$

In order to compute the cosine, we adopt the approach employed by Zobel and Moffat [18]. Therefore, the weight $w_{t,p.d}$ is computed as $w_{t,p.d} = 1 + ln(f_{t,p.d})$, where $f_{t,p.d}$ is the number of occurrences (frequency) of t in $p.d$; and the weight $w_{t,q.d}$ is obtained from the following formula $w_{t,q.d} = ln(1 + \frac{|P|}{df_t})$, where $|P|$ is the total number of documents in the collection. The *document frequency* df_t of a term t gives the number of documents in P that contains t. The higher the cosine value, the higher the textual relevance. The textual relevance is a value within the range $[0, 1]$ (property of cosine).

We also define the *impact* $\lambda_{t,d}$ of a term t in a document d, where d represents the description of an object $p.d$ or the query keywords $q.d$. The impact $\lambda_{t,d}$ is the normalized weight of the term in the document [1, 16], $\lambda_{t,d} = \frac{w_{t,d}}{\sqrt{\sum_{t \in d}(w_{t,d})^2}}$. The impact takes into account the length of the document and can be used to compare the relevance of two different documents according to a term t present in both documents. Consequently, the textual relevance $\theta(p.d, q.d)$ can be rewritten in terms of the impact [16], $\theta(p.d, q.d) = \sum_{t \in q.d} \lambda_{t,q.d} \cdot \lambda_{t,p.d}$.

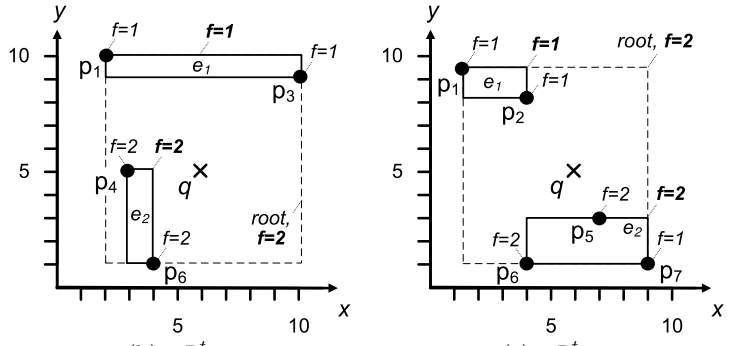
Other types of spatial proximity and textual relevance measures such as Okapi BM25 [13] can be supported by our framework. The focus of this paper is, however, on the efficiency of top-k spatial keyword queries. In the following, we present the S2I (Sect. 4) and describe the algorithms to process top-k spatial keyword queries efficiently (Sect. 5 and 6).

4 Spatial Inverted Index

The S2I was designed taking in account the following observations. First, terms with different document frequency should be stored differently. It is well-known

term	id	df_t	type	ptr	storage
bar	t_1	4	tree	↪	aR^{t_1}
pop	t_2	2	block	↪	$\langle p_1, p_5 \rangle$
pub	t_3	5	tree	↪	aR^{t_3}
rock	t_4	2	block	↪	$\langle p_2, p_3 \rangle$
samba	t_5	2	block	↪	$\langle p_4, p_7 \rangle$

(a) S2I.



(b) aR^{t_1}. (c) aR^{t_3}.

Fig. 2. Spatial Inverted index and the aR-tree of terms t_1 and t_3.

that the document frequency of terms in a corpus follows the Zipf's law, which means that there are a small number of terms that occur frequently, while most terms occur infrequently [10]. Current approaches that support top-k spatial keyword queries ignore this property and store frequent and infrequent terms in the same way. For example, the impact of a term that occurs in a single object has to be replicated in several nodes of the DIR-tree to indicate the path in the tree for the given object. Second, good support for distribution is important for scaling applications that can benefit from top-k spatial keyword queries. Third, keeping the index update is important. The current approaches require accessing one or several inverted files to perform a single update, which has a significant cost. Finally, response time is critical for several applications. Our approach accesses less disk pages and can perform queries more efficiently.

The S2I maps each term t to an aggregated R-tree (aR-tree) or to a block that stores the spatio-textual objects p that contain t. The most frequent terms are stored in aR-trees, one tree per term. The less frequent terms are stored in blocks in a file, one block per term. Similarly to a traditional inverted index, the S2I maps terms to objects that contain the term. However, we employ two different data structures, one for less frequent terms and another for more frequent terms that can be accessed in decreasing order of keyword relevance and spatial proximity efficiently.

The S2I consists of three components: vocabulary, blocks, and trees.

- **Vocabulary.** The vocabulary stores, for each distinct term, the number of objects in which the term appears (df_t), a flag indicating the type of storage used by the term (block or tree), and a pointer to a block or aR-tree that stores the objects containing the given term.
- **Blocks.** Each block stores a set of objects, the size of a block is an application parameter. For each object, we store the object identification $p.id$, the object location $p.l$, and the impact of term t in $p.d$ ($\lambda_{t,p.d}$). The objects stored in a block are not ordered.
- **Trees.** The aggregated R-tree [14] of a term aR^t follows the same structure of a traditional R-tree. A leaf-node stores information about the data objects: $p.id$, $p.l$, and $\lambda_{t,p.d}$. An intermediary-node stores for each entry (child node) a *minimum bounding rectangle* (MBR) that encloses the spatial location of all objects in the sub-tree. Differently from an R-tree, the nodes of an aR-tree store also an aggregated non-spatial value among the objects in its sub-tree. In our case, the aggregated value is the maximum impact of the term t among the objects in the sub-tree of the node. Hence, we can access the objects in aR^t in decreasing order of term relevance and spatial proximity.

Example 1. Fig. 2 presents the S2I created from the objects depicted in Fig. 1. In order to simplify the presentation, we assume in all examples that the impact of a term in a document ($\lambda_{t,d}$) is defined by the number of occurrences of the term in a document. We also assume that $\alpha = 0.5$ in all examples. We drop these assumptions in the experimental evaluation. The less frequent terms are stored in a block, while the more frequent terms are stored in an aR-tree, see Fig. 2(a). The aR-tree of terms t_1 and t_3 is depicted in Fig. 2(b) and 2(c), respectively. The root of R^{t_1} contains two entries e_1 and e_2. The entry e_1 contains the spatio-textual objects $\{p_1, p_3\}$, while e_2 contains $\{p_4, p_6\}$. The objects p_1 and p_3 have one occurrence of t_1 ($f = 1$), while p_4 and p_6 have two occurrences of t_1 ($f = 2$). The number of occurrences of a term in bold present the maximum number of occurrences of a term among the entries of each node.

The S2I has several good properties. First, terms with different document frequency are stored differently. Second, S2I has good support for distribution. S2I employs a different data structure for each term and can benefit from techniques used by traditional distributed search engines such as term partitioning [18]. Third, S2I provides good support for updates. Although one tree or block has to be accessed for each term in an object, the operations executed at each tree or block can be performed efficiently. Furthermore, the average number of distinct terms per document is small in most of the applications that are target of this query, as it can be seen in Table 2 (Section 7). Finally, S2I allows efficient query execution. For queries with a single keyword, only one small tree (in general) or a block needs to be accessed. For queries with several keywords only few nodes of a set of small trees or blocks are accessed. No access to external inverted indexes is required. In the following, we present the algorithm to process single-keyword (Sect. 5) and multiple-keyword queries (Sect. 6) employing S2I.

Algorithm 1 *SKA(MaxHeap H^t, Query q)*

```
1: INPUT: MaxHeap H^t with entries e in decreasing order of score τ(e,q) and the
   query q.
2: OUTPUT: The next object p with highest score τ(p,q).
3: Entry e ← H^t.pop()                    //e is the entry with highest score τ(e,q) in H^t
4: while e is not an object do
5:    if e is an intermediary-node then
6:       for each node n ∈ e do
7:          H^t.insert(n, τ(n,q))
8:       end for
9:    else                                                          //e is a leaf-node
10:      for each spatio-textual object p ∈ e do
11:         H^t.insert(p, τ(p,q))
12:      end for
13:   end if
14:   e ← H^t.pop()
15: end while
16: return e
```

5 Single-Keyword Queries

Top-k spatial keyword queries with a single keyword t can be efficiently processed using the S2I, since only one single block or tree containing the objects with the term t is accessed. If the objects are stored in a block, the query processing steps are: 1) retrieve all objects p in the block, 2) insert the objects in a heap in decreasing order of $\tau(p, q)$, and 3) report the top-k best objects. If the objects are stored in an aR-tree, we employ an incremental algorithm (Algorithm 1) to retrieve the objects in decreasing order of $\tau(p, q)$.

The SKA (Single Keyword Algorithm, Algorithm 1) visits entries e (nodes or objects) in an aR-tree in decreasing order of $\tau(e, q)$. The score of a node n, $\tau(n, q)$, and the score of an object p, $\tau(p, q)$, are computed in a similar way. The spatial proximity $\delta(n, q)$ is obtained computing the minimum Euclidean distance between q and any border of the MBR of n. On the other hand, the textual relevance $\theta(n, q)$ is obtained multiplying the impact of t in the query $\lambda_{t,q}$ by the impact of t in the node $\lambda_{t,n}$ that is the maximum impact among any entry e in the sub-tree of n. Consequently, the score $\tau(n, q)$ is an upper bound score for any entry e in the sub-tree of n, since any entry e in the sub-tree of n has a distance to q longer or equal $d(q, n)$, $d(q, n) \leq d(q, e)$; and the term impact of n is higher or equal the term impact of any entry e, $\lambda_{t,n} \geq \lambda_{t,e}$.

SKA receives as parameter a priority queue (MaxHeap) H^t and a query q. The heap stores the entries e of R^t in decreasing order of score $\tau(e, q)$. Initially, the heap has the root of R^t. The entry e (line 3) is the entry with highest score in H^t. If e is an intermediary-node (line 5), the algorithm computes the score of all nodes n children of e and insert n in H^t in decreasing order of $\tau(n, q)$ (lines 6-8). If e is a leaf-node (line 9), the algorithm computes the score of all objects p children of e and insert p in H^t in decreasing order of score (lines 10-12).

This process repeats until an object p achieves the top of the heap (line 4). This means that there is no other object in H^t with higher score than p. Therefore, p can be reported incrementally (line 16). The algorithm keeps the state of the variables for future access.

Example 2. Assume that we want to obtain the top-1 object among the objects depicted in Fig. 1 according to the query keyword $q.d = \{\text{bar}\}$ and the query location $q.l = (6, 5)$. The query processing starts accessing the S2I and acquiring the information that objects with the term "bar" (t_1) are stored in aR^{t_1}, Fig. 2(b). The SKA algorithm starts with the root of aR^{t_1} in H^{t_1} and inserts the entries e_1 and e_2 in the heap, $H^{t_1} = \langle e_2, e_1 \rangle$. The entry e_2 is in the top of H^{t_1} because it has a better score, $\tau(e_2, q) > \tau(e_1, q)$. The score of e_2 is higher because it has a higher term frequency ($f_{e_2,t_1} = 2$) and is nearer to q. The algorithm continues removing e_2 from the heap and inserting the objects p_4 and p_6 into $H^{t_1} = \langle p_4, p_6, e_1 \rangle$. In the next iteration, p_4 reaches the top of the heap H^{t_1} and is returned as top-1.

The problem of retrieving the objects in increasing order of score is similar to the problem of retrieving the nearest neighbors incrementally [8]. In the following, we present the algorithm to process multiple keyword queries.

6 Multiple-Keyword Queries

We divide this section in two parts. First, we define *partial-score* that is the score of an object according to a single term in the query, Sect. 6.1. Second, we present the algorithm to aggregate the objects retrieved in terms of partial-score to compute the top-k results, Sect. 6.2.

6.1 Partial-Score on Keyword

Processing multiple-keyword queries in the S2I requires aggregating objects from different *sources* S_i (aR-trees or blocks), where i refers to a distinct term $t_i \in q.d$. The naïve way is to retrieve objects p from each source S_i in decreasing order of score $\tau(p, q)$ (Equation 1) replacing $q.d$ by a query term $t \in q.d$, $\tau(p, \{t\})$. The final score is obtained adding the scores retrieved for each term. However, $\tau(p, q) \neq \sum_{t \in q.d} \tau(p, \{t\})$, because the spatial proximity is replicated in the scores computed for each term $\tau(p, \{t\})$. Hence, we propose to derive a partial-score on keyword $\tau^t(p, q)$ such that the score $\tau(p, q)$ can be computed in terms of the sum of partial-scores obtained from each source, $\tau(p, q) = \sum_{t \in q.d} \tau^t(p, q)$.

In order to obtain the partial-score $\tau^t(p, q)$, we rewrite Equation 1 so that the score $\tau(p, q)$ can be obtained in terms of the *sum* of partial-scores of t:

$$\begin{aligned}
\tau(p, q) &= \alpha \cdot \delta(p.l, q.l) + (1 - \alpha) \cdot \theta(p.d, q.d) \\
&= \alpha \cdot \delta(p.l, q.l) + (1 - \alpha) \cdot \sum_{t \in q.d} \lambda_{t,p.d} \cdot \lambda_{t,q.d} \\
&= \sum_{t \in q.d} (\alpha \cdot \frac{\delta(p.l, q.l)}{|q.d|} + (1 - \alpha) \cdot \lambda_{t,p.d} \cdot \lambda_{t,q.d})
\end{aligned}$$

where $|q.d|$ is the number of distinct terms in the query. From the equation above, we define *partial-score* $\tau^t(p,q)$ of p in relation to a term t as:

$$\tau^t(p,q) = \alpha \cdot \frac{\delta(p.l, q.l)}{|q.d|} + (1-\alpha) \cdot \lambda_{t,p.d} \cdot \lambda_{t,q.d} \ . \tag{4}$$

The partial-score reduces the weight of the spatial proximity $\delta(p.l, q.l)$ according to the number of distinct terms in the query $q.d$. Furthermore, once we have obtained an object p from a source S_i, we can also derive a lower bound partial-score $\tau^t_-(p,q)$ for p on the other sources in which p has not been seen yet, $\tau^t_-(p,q) = \alpha \cdot \frac{\delta(p.l,q.l)}{|q.d|}$. This is the lowest possible partial-score that p may have in a source where it has not been seen yet, since the proximity between p and q will not change. With the partial-scores $\tau^t(p,q)$ and $\tau^t_-(p,q)$, we can compute the lower bound and upper bound scores based on partial information about the object.

Example 3. Assume a query with two terms t_i and t_j, where the partial-score of p according to term t_i is known. The lower bond score of p, p_-, can be computed adding the partial-score of p according to t_i with the lower bound partial-score of p according to t_j, $p_- = \tau^{t_i}(p,q) + \tau^{t_j}_-(p,q)$.

In the following, we present the MKA algorithm that employs the partial-scores to compute the top-k results for queries with multiple-keywords.

6.2 Multiple Keyword Algorithm

The Multiple Keyword Algorithm (MKA) computes a top-k spatial keyword query progressively by aggregating the partial-scores of the objects retrieved for a given keyword. The objects containing a given term are retrieved in decreasing order of partial-scores from the source (aR-tree or block). In order to retrieve the objects in decreasing order of partial-score, we employ the SKA algorithm (Algorithm 1) replacing the score $\tau(p,q)$ by partial-score $\tau^t(p,q)$.

Each time an object p is retrieved from a source S_i ($S_i.next()$), we compute the lower bound score of p, update the upper bound score for any unseen object, and check if there is an object whose the lower bound score is higher or equal the upper bound of any other object. Those objects are reported progressively. We repeat this process until k objects have been found.

Algorithm 2 presents the MKA algorithm. MKA receives as parameter a top-k spatial keyword query q and reports the top-k results incrementally. MKA employ one source S_i for each distinct term $t_i \in q.d$ (line 3). Next, for each source S_i, the algorithm sets an upper bound u_i^- that maintains the highest partial-score among the objects unseen from S_i (line 6). The upper bound is updated every time that an object p is retrieved from S_i (line 10).

During each iteration (lines 7-23), MKA selects a source i (line 8), where the procedure $selectSource(q.d)$ defines the strategy to select the source. In this paper, we employ a round-robin strategy that selects a source S_i that is

Algorithm 2 *MKA(Query q)*

1: **INPUT:** The top-k spatial keyword query q.
2: **OUTPUT:** Progressively reports the top-k objects found.
3: $S_i \leftarrow$ *source* of the term $t_i \in q.d$
4: $C \leftarrow \emptyset$ *//List of candidate objects.*
5: $L_i \leftarrow \emptyset$ *//List of objects seen in the source* S_i
6: $u_i^- \leftarrow \infty$ *//Upper-bound score for any* $p \in S_i$
7: **while** $\exists S_i$ such that $S_i \neq \emptyset$ **do**
8: $i \leftarrow selectSource(q.d)$
9: $p \leftarrow S_i.next()$ *//Next object* p *in* S_i *with highest* $\tau^{t_i}(p,q)$
10: $u_i^- \leftarrow \tau^{t_i}(p,q)$ *//Update upper bound score for source* S_i
11: $L_i \leftarrow L_i \cup p$
12: $p_- \leftarrow \sum_{\forall j:p\in L_j} \tau^{t_j}(p,q) + \sum_{\forall j:p\notin L_j} \tau_-^{t_j}(p,q)$ *//Update lower bound score of* p
13: **if** $p \notin C$ **then**
14: $C \leftarrow C \cup p$
15: **end if**
16: **for each** $p \in C$ **do** *//Update upper bound score of the candidates*
17: $p^- \leftarrow \sum_{\forall j:p\in L_j} \tau^{t_j}(p,q) + \sum_{\forall j:p\notin L_j} max(u_j^-, \tau_-^{t_j}(p,q))$
18: **end for**
19: **while** $\exists p \in C : p_- \geq max_{\forall o\in C, o\neq p}(o^-)$ **do**
20: $C \leftarrow C - p$
21: reports p as next top-k, halt if $q.k$ objects have been reported
22: **end while**
23: **end while**
24: **if** less than $q.k$ objects have been reported **then**
25: return the objects in C with highest lower bound score p_-
26: **end if**

not empty. Other more sophisticated strategies such as keeping an indicator that express the effectiveness of selecting a source [6] can also be employed. MKA continues retrieving from S_i the next object p with highest partial-score $\tau^{t_i}(p,q)$ (line 9), and updating the upper bound score u_i^- (line 10) with the new partial-score retrieved from S_i. At this point, any unseen object in S_i has a lower or equal partial-score than u_i^-. Next, MKA marks that p has been seen in S_i adding p into L_i (line 11), and updating the lower bound score for p (line 12). The lower bound score is computed by summing the partial-scores known for p with the worst possible partial-score that p may have based on the sources in which p has not been seen yet. Then, MKA checks if p is in the candidate set C, inserting p otherwise (lines 13-15). Next, MKA updates the upper bound score for any object in the candidate set C (lines 16-18). The upper bound score for a given object in a source that it has not been yet, does not decrease bellow its lower bound score on that source (line 17). The objects $p \in C$ whose lower bound p_- is higher or equal the upper bound o^- of any other object in C (lines 19-22) are reported incrementally. MKA repeats this process until k objects has been reported, or the sources are empty. If the sources are empty and less than

Steps	aR^{t_1}	aR^{t_3}	Candidate set $\mathbf{C} = \{\mathbf{p}[\mathbf{p}_-, \mathbf{p}^-]\}$
	↓	↓	
2	$\tau^{t_1}(\mathbf{p_4}, q) \approx 1.2$	$\tau^{t_3}(\mathbf{p_5}, q) \approx 1.21$	$p_4[1.39, 2.41], p_5[1.42, 2.41]$
2	$\tau^{t_1}(\mathbf{p_6}, q) \approx 1.17$	$\tau^{t_3}(\mathbf{p_6}, q) \approx 1.17$	$p_4[1.39, \mathbf{2.37}], p_5[1.42, \mathbf{2.38}], p_6[2.34, 2.34]$
1	$\tau^{t_1}(\mathbf{p_3}, q) \approx 0.65$		$p_4[1.39, 2.37], p_5[1.42, \mathbf{1.86}], p_6[2.34, 2.34]$
1		$\tau^{t_3}(\mathbf{p_2}, q) \approx 0.68$	$p_4[1.39, \mathbf{1.88}], p_5[1.42, 1.86], p_6[2.34, 2.34]$

Fig. 3. State of some variables during the execution of MKA. The candidates p_2 and p_3 have low score and are omitted from the list of candidates.

k objects have been returned, MKA reports the objects in C with highest lower bound score (lines 24-26) as the remaining top-k.

Example 4. Assume a top-k spatial keyword query q, where $q.l = (5, 6)$, $q.d = \{bar, pub\}$, and $q.k = 1$. The MKA accesses the aR-trees aR^{t_1} (bar) and aR^{t_3} (pub) as depicted in Fig. 2. The state of some variables during the execution of MKA is shown in Fig. 3. After the second iteration (step), MKA has retrieved p_4 and p_5 from aR^{t_1} and aR^{t_3} respectively. MKA continues retrieving p_6 from both aR-trees aR^{t_1} and aR^{t_3}. Although p_6 has been found in both aR-trees, it cannot be reported as top-1 since the upper bound score of the other candidates p_4 and p_5 is still smaller than the score of p_6 (Fig. 3). However, after retrieving p_3 from aR^{t_1} and p_2 from aR^{t_3}, p_6 can be reported progressively as top-1 since its score is smaller than the upper bound score of the other candidates.

Aggregating term-scores to compute the top-k spatio-textual objects is similar to aggregating ranked inputs to compute the top-k results [9, 15]. For simplicity, we omit from the description of Algorithm 2 some implementation details that permit reducing the size of the candidate set and the number of comparisons to update the upper bound score [12]. In the following, we evaluate the performance of SKA and MKA algorithms.

7 Experimental Evaluation

In this section, we compare our approach the S2I against the DIR-tree proposed by Cong *et al.* [4]. All algorithms were implemented in Java using the XXL library[1]. The nodes of the aR-trees, employed in S2I, have a block size of 4KB that is able to store between 42 and 85 entries. The blocks in the file used to store the non-frequent items has a maximum size of 4KB that permits to store a maximum of 146 entries. The intuition behind this choice is that we store objects in an aR-tree only when there are enough objects to fill more than one node of an aR-tree. Each node of a DIR-tree also has a block size of 4KB and is able to store between 46 and 92 entries. The parameter β used to balance textual similarity and spatial location during the construction of the DIR-tree was set to 0.1 as suggested by Cong *et al.* [4].

[1] http://dbs.mathematik.uni-marburg.de/Home/Research/Projects/XXL

Parameter	Values
Number of results (k)	**10**, 20, 30, 40, 50
Number of keywords	1, 2, **3**, 4, 5
Query preference rate α	0.1, **0.3**, 0.5, 0.7, 0.9
Twitter dataset	1M, **2M**, 3M, 4M
Other datasets	Data1, Wikipedia (Wiki), Flickr, OpenStreetMap (OSM)

Table 1. Settings used in the experiments. The default values are presented in bold.

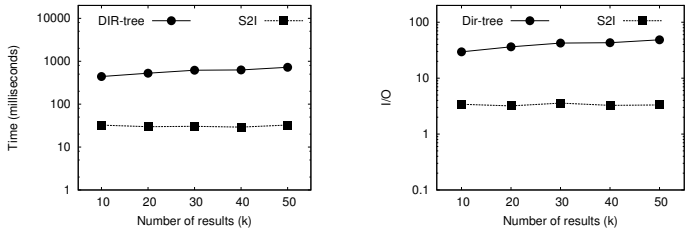
Datasets	Tot. no. of. objects	Avg. no. of unique words per object	Tot. no. of unique words	Tot. no. of words
Twitter1	1,000,000	11.94	553,515	12,542,414
Twitter2	2,000,000	12.00	1,009,711	25,203,367
Twitter3	3,000,000	12.26	1,391,171	38,655,751
Twitter4	4,000,000	12.27	1,678,451	51,661,462
Data1	131,461	131.70	101,650	32,622,168
Wikipedia	429,790	163.65	1,871,836	169,365,635
Flickr	1,471,080	14.49	487,003	25,417,021
OpenStreetMap	2,927,886	8.76	662,334	31,526,352

Table 2. Characteristics of the datasets.

Setup. Experiments were executed on a PC with a 3GHz Dual Core AMD processor and 2GB RAM. In each experiment, we execute 100 queries to warm-up the buffers, and collect the average results of the next 800 queries. The queries are randomly generated using the same vocabulary and the same spatial area of the datasets as used by Cong *et al.* [4]. We employed a buffer whose size was fixed in 4MB for both approaches. In the experiments, we measured the total execution time (referred as response time) and the number of I/Os (page faults). All charts are plotted using a logarithmic scale on the y-axis. The main parameters and values used through the experiments are presented in Table 1.

Datasets. Table 2 shows the characteristics of the datasets used in the experiments. We employed four Twitter datasets of 1M, 2M, 3M, and 4M objects each, where each object is composed by a Twitter message (tweet) and a random location where latitude and longitude are within the range [0,100]. In order to create these datasets, we used the first 10 million non-empty tweets from the Standford Twitter dataset[2]. The Data1 dataset was created combining texts from 20 Newsgroups dataset[3] and locations from LA streets[4]. This dataset is similar to the Data1 dataset used by Cong *et al.* [4]. However, instead of selecting only 5 groups, we employed all documents in the 20 Newsgroups dataset. We also conducted experiments on real datasets: Wikipedia, Flickr, and Open-

[2] http://snap.stanford.edu/data/twitter7.html
[3] http://people.csail.mit.edu/jrennie/20Newsgroups
[4] http://barcelona.research.yahoo.net/websmapm/datasets/uk2007



(a) Response time. (b) I/O.

Fig. 4. Response time and I/O varying the number of results (k).

StreetMap. The Wikipedia dataset is composed by Wikipedia articles with a spatial location. The Flickr dataset contains objects referring to photos taken in the area of London. Each object is composed by a spatial location and a text describing the photo (title and description). Finally, the OpenStreetMap[5] dataset contains objects downloaded from OpenStreetMap covering the entire planet.

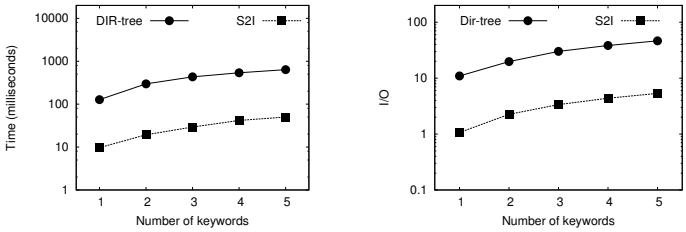
7.1 Query Processing Performance

In this section, we evaluate the query processing performance of the S2I and DIR-tree for different setups. In all experiments, we employ the default settings (Table 1) to study the impact on I/O and response time, while varying a single parameter.

Varying the number of results (k). Fig. 4 plots the response time and I/O of the S2I and DIR-tree, while varying the number of results k. The response time achieved using S2I is one order of magnitude better than using DIR-tree (Fig. 4(a)). Furthermore, the advantage of S2I increases when the number of results increases. The main reason for this is that S2I accesses less disk pages to process a query (Fig. 4(b). In order to process a query employing DIR-tree, the inverted files at each node are accessed to obtain the posting lists of each distinct keyword in the query. For example, a query with 3 distinct keywords is performed in two steps: first, the postings lists of each keyword is retrieved in order to identify the entries of the node that can contribute to the query results, then the relevant entries are visited in decreasing order of score. Although the size of the posting lists are small, since they are bounded by the maximum capacity of a node, the process to perform such queries on inverted indexes incurs in non-negligible cost.

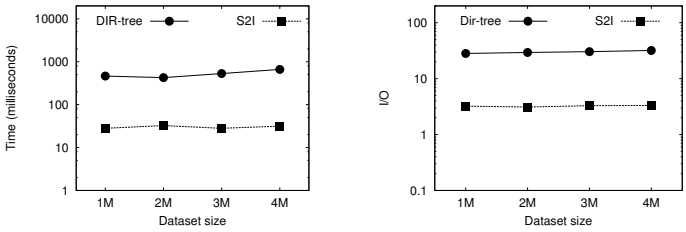
Varying the number of keywords. Fig. 5 depicts the response time and I/O, while varying the number of query keywords. Again, the response time (Fig. 5(a)) and I/O (Fig. 5(b)) achieved by using the S2I are one order of magnitude better. Single-keyword queries are processed efficiently employing the SKA algorithm. Hence, few pages are accessed during the query processing. As

[5] http://www.openstreetmap.org



(a) Response time. (b) I/O.

Fig. 5. Response time and I/O varying the number of keywords.



(a) Response time. (b) I/O.

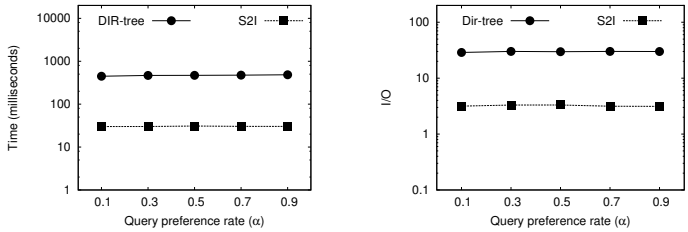
Fig. 6. Response time and I/O varying the cardinality of the Twitter dataset.

expected, the larger the number of keywords, the higher the I/O required to process the query. However, the advantage of S2I over DIR-tree remains constant, which demonstrates the efficiency of the MKA algorithm.

Varying the cardinality. In Fig. 6, we evaluate the impact of increasing the dataset size (cardinality) on response time and I/O. Again, the response time achieved by using the S2I is around one order magnitude better than S2I, see Fig. 6(a). Moreover, the advantage increases when the dataset increases. The same behavior is noted in the number of I/Os required, see Fig. 6(b).

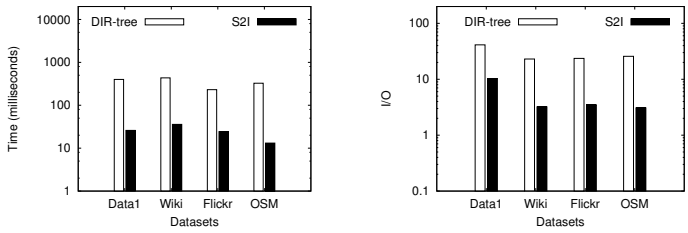
Varying the query preference parameter (α). In Fig. 7, we study the impact of α (Equation 1) on response time and number of I/Os. The performance of the S2I increases for higher values of the query preference parameter α (Fig. 7(a)), which means that S2I can terminate earlier if the preference parameter gives more weight to proximity over text relevance. The same behavior repeats in Fig. 7(b) that plots I/O. The query preference parameter does not present an impact on the performance of DIR-tree. The main reason for this is that the Twitter dataset has a large vocabulary and the objects are uniformly spread in the spatial space, which reduces the capacity of DIR-tree to put objects with similar content and similar location in the same node.

Varying the datasets. In Fig. 8, we present response time and I/O for different datasets. The S2I presents better response time (Fig. 8(a)) and I/O (Fig. 8(b)) for all datasets. The response time is influenced by the size of the dataset. The



(a) Response time. (b) I/O.

Fig. 7. Response time and I/O varying the query preference rate (α).



(a) Response time. (b) I/O.

Fig. 8. Response time and I/O for different datasets.

Wikipedia dataset is one of the largest dataset evaluated, since the text associated with each object has many keywords. On the other hand, Flickr and OpenStreetMap datasets have similar number of distinct terms per object, which is, in general, small. Consequently, the performance on those datasets is similar. Finally, Data1 presents the highest I/O. This happens because the total number of unique words in Data1 is small compared to its dataset size. Data1 is created combining 131,461 locations with only 20 thousand documents to create a dataset with 131,461 spatio-textual objects. Hence, there are many objects with the same textual description.

7.2 Maintenance and Space Requirements

In this section, we evaluate the maintenance cost and space requirements in both the DIR-tree and S2I.

Maintenance. We evaluate the cost of insertions, which are more frequent than updates and deletions. In order to obtain the insertion cost, we inserted 100 objects in S2I and DIR-tree and collected the average time. After each insertion, we flush the data in the index structures. The results are presented in Fig. 9.

Several storage units (blocks and trees) of the S2I, one per distinct term, are accessed due to an insertion of a new object. However, the tasks executed in a block or in an aR-tree are performed efficiently since they do not require computing the textual similarity between the new object and the objects currently

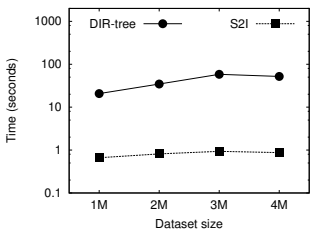


Fig. 9. Update time (seconds).

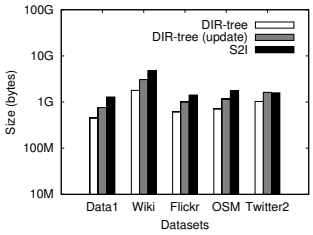


Fig. 10. Size of the indexes.

stored in the index. However, the cost of inserting a new object in a DIR-tree requires comparing the similarity between the text of the new object and the text of other several objects in order to find the region of DIR-tree that better accommodates the new object. Moreover, the insertion requires updating inverted files which is also challenging and costly.

Space requirements. Fig. 10 depicts the space required for both indexes DIR-tree and S2I. The size required by S2I is larger than the size required by DIR-tree. The main reason is that S2I employs more trees that do not use the space in the nodes effectively. The figure shows also the space required by DIR-tree to execute updates. This space includes the size of the vectors of the pseudo-documents at each node. The vectors are required only during updates, since at query time DIR-tree employs only inverted files. The space required by S2I to perform search and update is the same.

8 Conclusions

In this paper, we present a new index named Spatial Inverted Index (S2I) and algorithms (SKA and MKA) to support top-k spatial keyword queries efficiently. Similar to an inverted index, S2I maps distinct terms to the set of objects that contains the term. The list of objects that contain a term are stored differently according to the document frequency of the term. If the term occurs often in the collection, the objects with the term are stored in an aggregated R-tree and can be retrieved in decreasing order of partial-score efficiently. Differently, the objects of infrequent term are stored together in a block in a file. Furthermore, we present algorithms to process single-keyword (SKA) queries and multiple-keyword (MKA) queries efficiently. Finally, we show through extensive experiments that our approach outperforms the state-of-the-art approach in terms of query and update cost.

Acknowledgments

We are thankful to Massimiliano Ruocco for providing the Flickr dataset used in the experimental evaluation.

References

1. V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proc. of ACM Special Interest Group on Information Retrieval (SIGIR)*, pages 35–42, 2001.
2. N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM Int. Conf. on Management of Data (SIGMOD)*, pages 322–331, 1990.
3. Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *Proceedings of the ACM Int. Conf. on Management of Data (SIGMOD)*, pages 277–288, 2006.
4. G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. In *Int. Conf. on Very Large Data Bases (VLDB)*, pages 337–348, 2009.
5. I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *Proceedings of Int. Conf. on Data Engineering (ICDE)*, pages 656–665, 2008.
6. U. Güntzer, W. Balke, and W. Kießling. Optimizing Multi-Feature queries for image databases. In *Proceedings of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 419–428, 2000.
7. R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *Proceedings of the Int. Conf. on Scientific and Statistical Database Management (SSDBM)*, pages 1–10, 2007.
8. G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems (TODS)*, 24(2):265–318, 1999.
9. I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comp. Surveys*, 40(4):1–58, 2008.
10. T. Joachims. A statistical learning model of text classification for support vector machines. In *Proc. of ACM Special Interest Group on Information Retrieval (SIGIR)*, pages 128–136, 2001.
11. Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang. IR-tree: An efficient index for geographic document search. *Proceedings of the IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 99(4):585–599, 2010.
12. N. Mamoulis, M. L. Yiu, K. H. Cheng, and D. W. Cheung. Efficient top-k aggregation of ranked inputs. *ACM Transactions on Database Systems (TODS)*, 32(3):19, 2007.
13. C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
14. D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP operations in spatial data warehouses. In *Proceedings of the Int. Symposium on Advances in Spatial and Temporal Databases (SSTD)*, pages 443–459, 2001.
15. J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Efficient processing of top-k spatial preference queries. *Proceedings of the VLDB Endowment (PVLDB)*, 4(2):93–104, 2010.
16. G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.
17. Y. Zhou, X. Xie, C. Wang, Y. Gong, and W. Ma. Hybrid index structures for location-based web search. In *Proceedings of Int. Conf. on Information and Knowledge Management (CIKM)*, pages 155–162, 2005.
18. J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comp. Surveys*, 38(2):1–56, 2006.