Efficient and Scalable Method for Processing Top-k Spatial Boolean Queries

Ariel Cary [1], Ouri Wolfson [2], Naphtali Rishe [1]

[1] School of Computing and Information Sciences
Florida International University, Miami, FL 33199, USA
{acary001,rishen}@cis.fiu.edu

[2] Department of Computer Science,
University of Illinois at Chicago, Chicago, IL 60607, USA
wolfson@cs.uic.edu

Abstract. In this paper, we present a novel method to efficiently process *top-k* spatial queries with conjunctive Boolean constraints on textual content. Our method combines an R-tree with an inverted index by the inclusion of spatial references in posting lists. The result is a disk-resident, dual-index data structure that is used to proactively prune the search space. R-Tree nodes are visited in best-first order. A node entry is placed in the priority queue if there exists at least one object that satisfies the Boolean condition in the subtree pointed by that entry. We show via extensive experimentation with real spatial databases that our method has increased performance over alternate techniques while scaling to large number of objects.

1 Introduction

Today's Internet applications typically offer users the ability to associate geographical information to Web content, a process known as "geotagging". For example, Wikipedia has standardized geotagging of their encyclopedia articles and images via templates [6]. Furthermore, technological advances in digital cameras and mobile phones allow users to acquire and associate geospatial coordinates, via built-in GPS devices or Wi-Fi triangulation, to media resources. Additionally, Web content can be automatically paired with geographical coordinates, for instance, exploiting content features, such as place names or street addresses, in combination with gazetteers. Thus, the powerful combination of Internet applications, GPS-enabled devices, and automatic geotagging can potentially generate large amounts of georeferenced content. On the structured end, spatial databases usually contain rich textual descriptions, stored in non-spatial attributes. For example, a database of property parcels may store property's owner name, description, and street address in addition to its coordinates.

A key problem recently tackled by the academia and industry is spatial searches with text constraints in geographical collections [3] [7] [11] [9] [10]. For example, in a database of parcels, we may be interested in finding nearby houses to Miami Beach (spatial constraint) that have backyard and are located on Collins Avenue (text constraint). Typically, query keywords are assumed to be conjunctively connected. That is, records containing all query keywords are retrieved. In the general case, text constraints may involve complex combinations of keywords with logical connectives beyond the conjunctive semantics. For instance, in the database of parcels, fire fighters traveling in a truck may want to quickly determine the nearest parcels that have swimming pool and are *not* located in buildings for water replenishment in an emergency.

As geospatial collections increase in size, the demand of efficient processing of spatial queries with text constraints becomes more prevalent. In this paper, we propose a method for efficiently processing top-k nearest neighbor queries with text constraints where keywords are combined with the three basic Boolean operators AND, OR, and NOT. Our method uses an R-tree to guide the spatial search and an inverted file for text content retrieval, which are combined in a novel hybrid spatial–keyword index. The specific contributions of this paper are:

1. We define a *top-k* spatial Boolean (*k-SB*) query that finds nearest neighbor objects satisfying Boolean constraints on keywords combined with conjunctive ($\wedge$), disjunctive ($\vee$), and complement ($\neg$) logical operators.
2. We propose a novel hybrid Spatial-Keyword Index (*SKI*) to efficiently process *k-SB* queries. A salient feature of *SKI* is that it only searches subspaces that do contain objects satisfying the query Boolean predicate.
3. We execute extensive experimentation on an implementation of our method over large spatial databases. Experimental results show that the proposed method has excellent performance and scalability.

Section 2 discusses related work to our research. Section 3 formally defines the problem. Section 4 presents the proposed hybrid indexing approach and query processing algorithms. Experimental study on an implementation of our hybrid index is conducted in Section 5. Section 6 presents our concluding remarks.

2 Related Work

The R-tree traversal method in our work is inspired in Hjaltason and Samet's [5] incremental top-k nearest neighbor algorithm using R-trees [1]. Performance improvements on the original R-tree work have been proposed, e.g. R*-tree [13] , R+-tree [14], and Hilbert R-tree [15]. Any of these variants can replace the R-tree index used in the proposed hybrid spatial keyword index without modifying our search algorithms. In information retrieval, inverted files are arguably the most efficient index structure for free-text search [2] [12].

The problem of retrieving spatial objects satisfying non-spatial constraints has been studied in the recent past. Park and Kim [10] proposed RS-trees, a combination of R-trees and signature trees for attributes with controlled cardinality; signature chopping is suggested to mitigate *combinatorial errors* [8] (database overrepresentation) of superimposed signatures. Harinharan et al. [9] proposed to include a list of terms in every node of an R-tree. De Felipe et al. [11] augmented signature files in R-tree nodes with similar constraints as [10]. Recently, Cong et al. [3] augmented an inverted file in every node of an R-tree, and used a ranking function that combines spatial proximity and text relevancy. Our work differs in that we assume distance as ranking score, and we focus on efficiently processing Boolean constraints on textual data. Further, none of the previous works offer efficient processing of the complement logical operator, which limits their applicability to the *k-SB* queries we considered in this work. Likewise, modern Web search engines, like Google and Yahoo!, offer *Local Search* services. Advanced querying options are provided to include and exclude certain terms from the search result. These are similar to the *k-SB* queries we consider. However, specific search algorithms are kept confidential by their owning companies.

3 Problem Definition

A spatial database $D = \{o_1, o_2, ..., o_N\}$ is a set of objects such that every $o \in D$ has a pair of attributes $< p, T >$, where: $p \in E$ is a point in a metric space E with distance $dist(p_1, p_2)$, and $T = \{t_1, t_2, ...\}$ is a document as a set of terms.

A *top-k* spatial Boolean (*k-SB*) query Q is a triple $< l, k, B >$, where: $l \in E$ is the query location (*spatial constraint*), k is the desired output size, and B is the conjunctive Boolean predicate (*text constraint*). B is a set of keywords prefixed with Boolean operators $\{\wedge, \vee, \neg\}$, conjunctively connected as follows:

$$B = \Big[\wedge(A = \{a_1, a_2, ...\}) \bigwedge \vee(C = \{c_1, c_2, ...\}) \bigwedge \neg(G = \{g_1, g_2, ...\})\Big] \quad (1)$$

A (*AND*-semantics), C (*OR*-semantics), G (*NOT*-semantics) are subsets of terms prefixed with $\wedge$, $\vee$, and $\neg$, respectively. An object $o \in D$ *satisfies* B if:

$$[(\forall a \in A : o.T \cap a \neq \emptyset) \wedge (\exists c \in C : o.T \cap c \neq \emptyset) \wedge (\forall g \in G : o.T \cap g = \emptyset)] \quad (2)$$

The result of the *k-SB* query Q is the list:

$$L = \{o_i \in D, i = 1, ..., n_L | o_i \ satisfies \ B \wedge n_L \leq k\}, \text{ such that:}$$
$$\forall o \in (D \setminus L) \ : \ [dist(o.p, l) \geq arg\, max_{r \in L} dist(r.p, l) \vee \neg(o \ satisfies \ B)] \quad (3)$$

Objects in L are sorted by distance to l in non-decreasing order. In other words, a *k-SB* query Q returns the k nearest neighbor objects to the query location l that satisfy the conjunctive Boolean predicate B. In this work, we assume E is the Euclidean space. The problem is how to efficiently compute L.

Example: In database D_1 of Table 1, the query "Find *top-10* houses nearby Miami that have masterbed with bathtub, have a pool or backyard, and are not located in a building" translates to the following *k-SB* query:
$Q_1 = \{Miami, 10, [\wedge(masterbed, bathtub) \bigwedge \vee(pool, backyard) \bigwedge \neg(building)]\}$
and retrieves $L = \{o_3, o_8\}$.

Table 1. Property parcel database $D_1 = \{o_1, o_2, ..., o_{12}\}$. For every textual term (t), the list of objects containing t is shown.

Term	Object List	Term	Object List
backyard (t_1)	$\{o_2, o_3, o_6, o_8\}$	collins (t_4)	$\{o_2, o_6, o_{10}\}$
bathtub (t_2)	$\{o_3, o_5, o_8, o_9\}$	masterbed (t_5)	$\{o_3, o_8, o_{11}\}$
building (t_3)	$\{o_1, o_5, o_7, o_{12}\}$	miami (t_6)	$\{o_1, o_3, o_4, o_{10}\}$

4 Hybrid Spatial-Keyword Indexing

In designing the hybrid index, we pursue the following objectives. First, we want to attain fast retrieval even when matching objects are located far away from one another. Second, we want to efficiently filter objects not satisfying the query Boolean constraints on keywords. A key challenge is to perform small number of computations to eliminate as many non-candidate objects possible. In particular, NOT-semantics constraints may substantially shrink the output size, and lead to unnecessary scans. Third, we want to maintain low storage requirements while keeping high query performance. With these objectives in mind, our indexing approach leverages the strengths of R-trees [1] in spatial search, and modifies an inverted file [2] for efficient processing of Boolean constraints. The combination of indexing techniques yields a hybrid data structure: *Spatial-Keyword Index* (SKI). We next introduce two important definitions in *SKI*.

Definition 1. *Given an R-tree R with fanout m, a super node s is the list of m leaf (level-1) nodes that have the same parent node. The universe of super nodes of R is $S(R) = [s_1, s_2, ...]$, where s_1 references the left-most leaf nodes.*

Definition 2. *The term bitmap of term t at super node s is a fixed–length bit sequence $I(t, s)$ of size m^2, where the i–th bit is computed as follows:*

$$I(t,s)[i] = \begin{cases} 1 & \textit{if } s[i] \textit{ points to object } o : t \in o.T \\ 0 & \textit{otherwise} \end{cases} \tag{4}$$

For an R-tree with L levels, a *super node* s contains $O(m)$ leaf nodes, or equivalently $O(m^2)$ object pointers, and $|S(R)| = O(m^{(L-2)})$ for $L > 1$. A single-level R-tree has no super nodes. Figure 1 shows *super node* s_1 of an R-tree built on D_1, and $I(\text{"}miami\text{"}, s_1)$ term bitmap.

4.1 Spatial Keyword Index

The hybrid spatial keyword index (SKI) is composed of two building blocks:

a) R-tree Index (R): A modified R-tree built with spatial attributes of D. Entries in R's inner nodes are augmented with index ranges $[a, b]$, where s_a and s_b are the left-most and right-most, respectively, super nodes contained in the subtree rooted at the node entry. Ranges in leaf-node entries contain a single value, the index of the super node containing the leaf node.

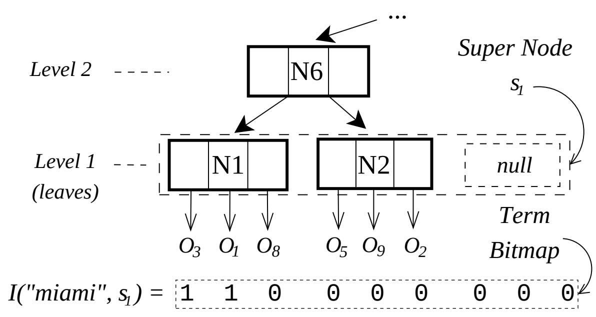

Fig. 1. Super node s_1 composed of leaf nodes $[N_1, N_2]$, and term bitmap for *"miami"*.

b) Spatial Inverted File (SIF): A modified inverted file constructed on the vocabulary $V = \left\{\bigcup_{o \in D} o.T\right\}$. The Lexicon contains terms in V and their document frequencies (df). Posting lists are modified to include spatial information from R. The posting of any term t contains a list of all its *term bitmaps* (rather than documents) sorted by super node index as follows:

$$Posting(t) = [I(t, s_1), I(t, s_2), ...] \text{ where } s_i \in S(R) \quad (5)$$

Efficiency Considerations We organize posting elements in a B+tree to allow fast random and range retrieval. Keys are $< t, i >$ pairs while values are bitmaps $I(t, s_i)$. Also, in order to reduce storage requirements, we compress $I(t, s_i)$ using the Word-Aligned Hybrid bitmap compression method (WAH) [4]. WAH method allows fast bitwise computations with logical operators AND, OR, and NOT on uncompressed bitmaps, which is capitalized during query processing.

Figure 2 shows R and SIF structures for database D_1 of Table 1.

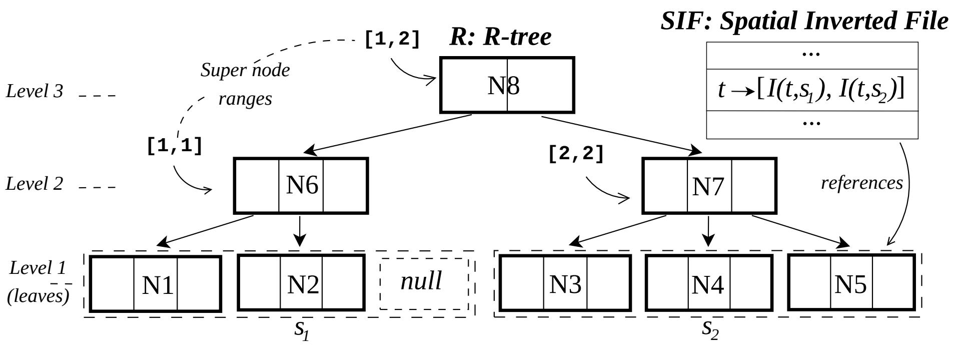

Fig. 2. Hybrid Spatial-Keyword Index for database D_1 in Table 1.

4.2 Processing *k-SB* Queries

In order to process query $Q =< l, k, B >$, R-tree R is traversed from the root node following the best-first traversal algorithm proposed in [5]. That is, node entries are visited in order of proximity of their *minimum bounding rectangles* (MBR) to location l. A node entry e is placed in a priority queue, with priority equal to $dist(MBR, l)$, if at least one object within e's subtree satisfies Boolean predicate B. The structure SIF is used to qualify e. A powerful feature of the previous filter is that unnecessary subtree traversals are eliminated altogether.

Algorithm 1 shows the steps involved in processing *k-SB* queries using SKI. The algorithm starts by resorting query keywords by documents frequency (line 1) in such a way that as many object candidates as possible are eliminated with few posting list merges. For instance, infrequent terms have large number of $0s$ in their term bitmaps, and possibly short $Posting()$ lists, which are adequate to be processed first for AND-semantics terms. In line 2, the priority queue, result list, and a globally accessible hash map M are initialized. M caches merged term bitmaps of previously evaluated super nodes during query execution. Next, R is traversed in best-first order starting from its root in lines 4–9. Node entries e are evaluated w.r.t B by the function $isSubtreeCandidate$ (line 8). Only when e's subtree has at least one object that satisfies B is it pushed into the queue.

Algorithm 1: Process *k-SB* Query

```
   Input: k-SB query $Q =< l, k, B >$
   Output: A list of objects satisfying $Q$ (see Equation 3)
   begin
1      Sort term subsets in $B$ by document frequency ($df$) as follows:
         $A$ ($AND$) in ascending order, and $C, G$ ($OR, NOT$) in descending order
2      Initialize: priority $Queue \leftarrow R.root$; list $L \leftarrow \emptyset$; global hash map $M \leftarrow \emptyset$
3      $r \leftarrow 0$        /* number of B-satisfying objects retrieved so far */
4      while ($Queue \neq \emptyset$ and $r < k$) do
5          $Node\ n \leftarrow Queue.pop()$
6          if (n is obj. pointer) then  $r \leftarrow r + 1$
             $L.add(getObject(D, n))$                    /* retrieve $o \in D$ */
7          else  for (every entry e in node n) do
8              if ($isSubtreeCandidate(B, n, [e\text{'s position in } n])$) then
9                  $Queue.push(e.ptr)$ with priority $dist(e.MBR, l)$
10     return $L$
```

isSubtreeCandidate function, described in Algorithm 2, evaluates B predicate by merging query term bitmaps on a range of super nodes, one super node at a time, until one candidate is found (lines 4–9). This processing style is similar to Document-At-A-Time processing in inverted files [2], except that postings are not exhausted. Logical bitwise operations are performed on term bitmaps (lines 5–7) according to their semantics. Complement operator requires term bitmaps to be flipped (converting 1*s* into 0*s* and vice versa), which is accomplished by the *flip* function (line 7). Next, if the merged bitmap has at least one bit set (line 8), meaning there is a candidate, then it is cached in M (line 8), and the function returns true. Otherwise, B is evaluated at the next super node in the range until a candidate is found, or the range is exhausted. In the latter worst case, the subtree is discarded in its entirety. Since a super node references $O(m^2)$ objects, a range $[a, b]$ can potentially filter out $O(m^2 \times |a - b|)$ objects. The *I/O* cost is remarkably only $O(|B| \times \log(|V|) \times |a - b|)$, where $|B|$ is the number of query terms, $|V|$ the vocabulary size, and $\log(|V|)$ the cost of term bitmap retrievals from a B+tree (see Section 4.1).

Algorithm 2: isSubtreeCandidate

```
Input: $B$: query predicate; $n$: node; $i$: positional index
Output: True if $\exists$ $o$ that satisfies $B$ within subtree at $n[i]$, false otherwise
begin
1    if (n is leaf node) then
2        if (The i-th bit in $M(n[i].a)$ is set) then return true
3        else return false
4    else for ($j \leftarrow n[i].a$ to $n[i].b$) do
5        $pe \leftarrow \bigwedge_{t \in B.A} I(t, j)$                /* execute bitwise operations */
6        $pe \leftarrow pe \wedge [\bigvee_{t \in B.C} I(t, j)]$      /* on term bitmaps over */
7        $pe \leftarrow pe \wedge [\bigwedge_{t \in B.G} flip(I(t, j))]$   /* super node range in n[i] */
8        if ($cardinality(pe) > 0$) then $M.add(key = j, value = pe)$
9        return true
10   return false
```

5 Experiments

We conducted a series of querying experiments with three real spatial datasets explained in Table 2. Records contain geographical coordinates, and between 30 and 80 text attributes (concatenated in a term set). *SKI* was implemented in Java, and experiments ran on an Intel Xeon E7340 2.4GHz machine with 8GB of RAM. We measured average number of random *I/Os* and response times in processing *k-SB* queries and compared performance with two baselines:

Baseline 1 (IFC) An inverted file containing object coordinates in addition to object pointers. Queries are processed in two phases. First, term postings are merged according to B semantics. Second, satisfying objects are sorted by proximity to query location. The top-k objects in the sorted list are returned.
Baseline 2 (RIF) An R-tree with every node augmented with an inverted file on keywords within its subtree. This baseline is inspired by arts [3] [9]. At query

Table 2. Experimental spatial datasets. Dataset and vocabulary sizes are in millions.

D	$\|D\|$	$\|V\|$	Subject
FL	10.8	21.2	Property parcels in the Florida state.
YP	20.4	40.8	Yellow pages of businesses in the United States.
RD	23.0	64.8	Road segments in the United States.

time, R-tree nodes are visited in best-first order w.r.t. spatial attributes. B is evaluated with the inverted file at the node, except for NOT-semantics terms.

Workload Every vocabulary was sorted by document frequency (*df*) and divided in three quantiles: **S**: Terms with *df* < 1–quantile (infrequent terms), **M**: Terms with *df* < 2–quantile, **L**: Terms with *df* < 3–quantile (entire vocabulary). In each quantile, *k-SB* queries were composed by randomly picking between 3 and 8 terms and prefixing them with $\{\wedge, \vee, \neg\}$ operators to form B. k was fixed to 20.

Figures 3 shows the average number of I/Os and elapsed time over 50 *k-SB* queries of each workload type $\{S, M, L\}$ on every dataset. IFC shows performance advantage when query terms are relatively infrequent (S). Short posting lists can be quickly evaluated to compute query result, whereas SKI and RIF spend additional R-tree traversals. When query terms become more frequent (M and L), IFC incurrs in expensive long posting list merges, which is observed in peaks of Figure 3.a for L queries. RIF performs acceptably for S queries but degrades for M and L queries. This may be due to filtering limitations in R-tree upper levels. Eventually, subtrees known (via inverted file) to contain query terms are traversed. However, terms may belong to different objects, i.e. no single object satisfies B predicate. In the same vein, RIF must wait until objects are retrieved to apply NOT-semantics filters, which can also degrade its performance. In summary, we observed consistent enhanced retrieval performance using the proposed hybrid indexing and query processing methods.

6 Conclusions

In this paper, we proposed a disk-resident hybrid index for efficiently answering k-NN queries with Boolean constraints on textual content. We combined modified versions of R-trees and inverted files to achieve effective pruning of the search space. Our experimental study showed increased performance and scalability on large spatial datasets over alternate methods.

Acknowledgments. This research was supported in part by NSF grants IIS-0837716, CNS-0821345, HRD-0833093, IIP-0829576, IIP-0931517, DGE-0549489, IIS-0957394, and IIS-0847680.

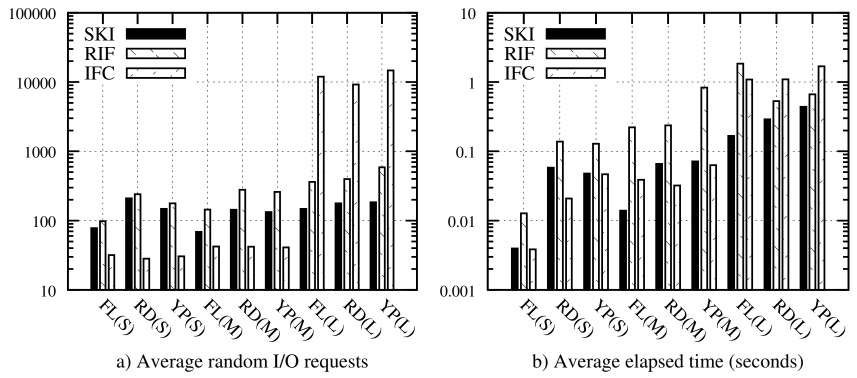

Fig. 3. Performance metrics on 50 *k-SB* query runs. *Y-axis* is in logarithmic scale.

References

1. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In SIGMOD, pp. 47–57. ACM, New York, 1984.
2. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Comput. Surv. 38 (2), 2006.
3. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. Proc. VLDB Endow. 2 (1), 337–348, 2009.
4. Wu, K., Otoo, E.J., Shoshani, A.: Optimizing bitmap indices with efficient compression. ACM Trans. Database Syst. 31 (1), 1–38, 2006.
5. Hjaltason, G., Samet, H.: Distance browsing in spatial databases. ACM Trans. Database Syst. 24 (2), 265–318, 1999.
6. WikiProject on geographical coordinates, `http://en.wikipedia.org/wiki/Wikipedia:WikiProject_Geographical_coordinates`
7. Xin, D., Han, J.: P-Cube: Answering preference queries in multi-dimensional space. In ICDE, pp. 1092–1100, IEEE Computer Society, 2008.
8. Chang, W.W., Schek, H.J.: A signature access method for the Starburst database system. In VLDB, pp. 145–153, 1989.
9. Hariharan, R., Hore, B., Li, C., Mehrotra, S.: Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In SSDBM, p. 16, 2007.
10. Park, D.J., Kim, H.J.: An enhanced technique for k-nearest neighbor queries with non-spatial selection predicates. Multimedia Tools and Apps., 79–103, 2004.
11. De Felipe, I., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In ICDE, pp. 656–665, IEEE Computer Society, 2008.
12. Zobel, J., Moffat, A., Ramamohanarao, K.: Inverted files versus signature files for text indexing. ACM Trans. Database Syst. 23 (4), 453–490, 1998.
13. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: An efficient and robust access method for points and rectangles. In SIGMOD, pp. 322–331, 1990.
14. Sellis, T.K., Roussopoulos, N., Faloutsos, C.: The R+-tree: A dynamic index for multi-dimensional objects. In VLDB, pp. 507–518, 1987.
15. Kamel, I., Faloutsos, C.: Hilbert R-tree: An improved R-tree using fractals. In VLDB, pp. 500–509, 1994.