World Wide Web (2013) 16:111–139
DOI 10.1007/s11280-012-0159-3

RASIM: a rank-aware separate index method for answering top-*k* spatial keyword queries

Hyuk-Yoon Kwon · Kyu-Young Whang · Il-Yeol Song · Haixun Wang

Received: 18 November 2011 / Revised: 18 February 2012 /
Accepted: 27 February 2012 / Published online: 19 May 2012

Abstract A top-*k* spatial keyword query returns *k* objects having the highest (or lowest) scores with regard to spatial proximity as well as text relevancy. Approaches for answering top-*k* spatial keyword queries can be classified into two categories: the separate index approach and the hybrid index approach. The separate index approach maintains the spatial index and the text index independently and can accommodate new data types. However, it is difficult to support top-*k* pruning and merging efficiently at the same time since it requires two different orders for clustering the objects: the first based on scores for top-*k* pruning and the second based on object IDs for efficient merging. In this paper, we propose a new separate index method called *Rank-Aware Separate Index Method (RASIM)* for top-*k* spatial keyword queries. RASIM supports both top-*k* pruning and efficient merging at the same time by clustering each separate index in two different orders through the partitioning technique. Specifically, RASIM partitions the set of objects in each index into *rank-aware (RA) groups* that contain the objects with similar scores and applies the first order to these groups according to their scores and the second order to the objects within each group according to their object IDs. Based on the RA groups, we propose two query processing algorithms: (i) *External Threshold Algorithm (External*

H.-Y. Kwon · K.-Y. Whang (✉)
Department of Computer Science, Korea Advanced Institute of Science
and Technology (KAIST), Daejeon, South Korea
e-mail: kywhang@cs.kaist.ac.kr

H.-Y. Kwon
e-mail: hykwon@mozart.kaist.ac.kr

I.-Y. Song
College of Information Science and Technology, Drexel University, Philadelphia, USA
e-mail: song@drexel.edu

H. Wang
Microsoft Research Asia, Beijing, China
e-mail: haixunw@microsoft.com

TA) that supports top-*k* pruning in the unit of RA groups and (ii) *Generalized External TA* that enhances the performance of External TA by exploiting special properties of the RA groups. RASIM is the first research work that supports top-*k* pruning based on the separate index approach. Naturally, it keeps the advantages of the separate index approach. In addition, in terms of storage and query processing time, RASIM is more efficient than the IR-tree method, which is the prevailing method to support top-*k* pruning to date and is based on the hybrid index approach. Experimental results show that, compared with the IR-tree method, the index size of RASIM is reduced by up to 1.85 times, and the query performance is improved by up to 3.22 times.

Keywords top-*k* spatial keyword query · separate index · top-*k* pruning

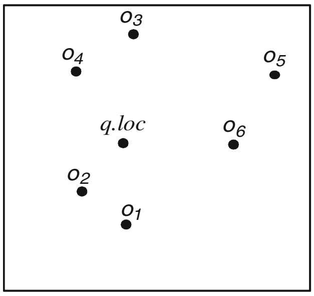
1 Introduction

Recently, there is a growing need of integrating databases and information retrieval (simply, the *DB-IR integration*) [8, 29] due to widespread use of the Web [16]. The emergence of mobile technologies in the Web [2, 22] adds the spatial dimension to DB-IR integration. Storing and managing various types of such complex data and integrated processing of queries on them are of growing importance. An example of such fusion is storing each spatial object along with a text description in the database and processing complex queries on them. Indeed, commercial Web search engines such as Google Maps support queries including both keywords and geographical information. Other applications that use such queries include map services, local searches, and local advertisements [10]. We call this type of query the *spatial keyword query* [15].

A *spatial keyword query* consists of a query region and a set of keywords, and returns the objects that are inside the query region and that include the keywords [15]. Efficient query processing is based on the spatial index and the text index. Existing work can be classified into two categories according to their index structures [15]: the separate index approach and the hybrid index approach. The former [9] maintains the spatial index and the text index independently, and finds the result objects by merging objects retrieved through the two indexes. The latter [15, 34] builds a single index that combines the spatial and text indexes.

A *top-k spatial keyword query* consists of a query point and a set of keywords, and returns *k* objects that have the highest (or lowest) scores. Here, the score combines spatial proximity between an object and the query point as well as text relevancy between the object and the keywords in the query [12]. Figure 1 shows a sample data set used in the running examples of this paper. We assume that each object consists of a location description and a text description: Figure 1a shows the location description of the objects; Figure 1b the text description. An example of top-*k* spatial keyword queries is as follows: "Find the top three objects whose location is closest to the query point *q.loc* and whose text description is most relevant to keywords 'vegetable' and 'food'."

Just like spatial keyword queries, existing work on top-*k* spatial keyword queries can be classified into two categories: the separate index approach and the hybrid index approach [21]. Martin et al. [21] compare the two approaches for top-*k* spatial



(a) Location description of objects.

Objects	Text Description
o_1	…vegetable...food...vegetable…food…food…
o_2	…meat…vegetable…
o_3	…vegetable…vegetable…food…food…
o_4	…vegetable….vegetable…vegetable…
o_5	…meat…vegetable…food…food…
o_6	…meat…food…

(b) Text description of objects.

Figure 1 A sample data set consisting of location description and text description.

keyword queries. The separate index approach is shown to be superior to the hybrid index approach because (i) processing nearest neighbor queries (queries with spatial predicates only) or text retrieval queries (queries with text predicates only) is more efficient; (ii) updates can be handled in each index independently; (iii) adding attributes with new data types or removing an existing one is easy.

Top-*k* spatial keyword query processing relies on two techniques: (i) efficient top-*k* pruning [9], i.e., stopping execution when the top-*k* results have been found; and (ii) efficient merging of objects retrieved from the two indexes. To support top-*k* pruning and efficient merging, two different orders of clustering objects are required: top-*k* pruning requires ordering objects based on their scores, and efficient merging requires ordering objects based on their object IDs. However, objects in an index are usually clustered in a single order.

Chen et al. [9] proposed a separate index method that supports efficient merging by clustering both the objects in the spatial index and those in the text index in the order of object IDs. But, this method does not handle top-*k* pruning. Cong et al. [10] and Li et al. [19] proposed the *IR-tree method* that supports top-*k* pruning using the hybrid index approach, which simultaneously considers spatial proximity and text relevancy of an object in the index structure. The IR-tree method is the prevailing method that supports top-*k* pruning for top-*k* spatial keyword queries to date.

In this paper, we propose a novel separate index method, called the *Rank-Aware Separate Index Method (RASIM)*, that supports both top-*k* pruning and efficient merging. RASIM is novel in the sense that objects in each index are clustered in two different orders, so that it supports top-*k* pruning and efficient merging simultaneously. This is achieved by a partitioning technique that partitions objects in each index into *rank-aware groups* that contain the objects with similar scores. We apply the first order to these groups based on their scores and the second order to the objects within each group based on their object IDs. Based on the rank-aware groups, we propose two query processing algorithms: *External Threshold Algorithm (External TA)* and *Generalized External TA*. External TA extends Threshold Algorithm (TA) proposed by Fagin et al. [11] to handle groups (mapped to disk pages) of objects rather than individual objects as the unit of pruning. Generalized External TA enhances the performance of External TA by exploiting special properties of the rank-aware groups to be explained in Section 4.3.3.

We make the following four contributions. First, RASIM is the first work that supports top-*k* pruning based on the separate index approach for top-*k* spatial keyword queries. Second, we propose two query processing algorithms based on the rank-aware groups: External TA and Generalized External TA. Third, RASIM is space efficient in that it supports top-*k* pruning without having to store part of the inverted index redundantly as opposed to the IR-tree method. Experimental results show that the size of RASIM is reduced by up to 1.85 times compared with that of the IR-tree method. Fourth, RASIM has better query performance than the IR-tree method. There are three reasons for this improvement: (i) the pruning power of RASIM, which follows that of TA, is comparable to or better than the IR-tree method; (ii) RASIM consolidates the index so that a single I/O is capable of retrieving objects in both orders, which reduces index access time, while the IR-tree method needs to access additional index to perform the same function; and (iii) RASIM clusters the text index in one file while the IR-tree method stores it in multiple files over different nodes resulting in multiple fragmented text indexes. Experimental results show that the query performance of RASIM is improved from 1.08 to 3.22 times compared with that of the IR-tree method.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 formally defines the top-*k* spatial keyword query. Section 4 proposes the rank-aware separate indexes and query processing algorithms based on them. Section 5 presents the results of performance evaluation. Section 6 summarizes and concludes the paper.

2 Related work

Top-k queries A *top-k query* returns *k* objects that have the highest (or lowest) scores according to user preference [18]. The representing work for top-*k* queries are Threshold Algorithm (TA) proposed by Fagin et al. [11] and the ONION method proposed by Chang et al. [7]. TA uses, as the input, multiple lists each of which is sorted by one attribute. The ONION method regards objects as points in a multidimensional space and constructs a list of layers for them.

Text retrieval queries A *text retrieval query* returns documents that include a set of keywords given in the query. The inverted index is the most widely used for this query type [3, 35]. Figure 2 shows the structure of the inverted index. The inverted index consists of keywords and posting lists with each list being associated with a keyword. Each posting list consists of postings where each posting contains the information such as the document identifier (*DocID*) and the frequency of occurrence of the keyword, i.e., *the term frequency* (*TF*), in the document. In addition, a B+-tree on keywords (simply, the *keyword index*) can be built to search the posting list of a specific keyword efficiently.

The representatives of ranking measures for text retrieval queries are the query-independent score such as PageRank [5] and the query-dependent score such as TF-IDF [9]. For the former, we can efficiently retrieve the top-*k* results by maintaining the postings in each posting list in the order of the (query-independent) score and by accessing them according to this order [5]. For the latter, since the cost of calculating the exact scores of all the documents with respect to the given keywords is expensive

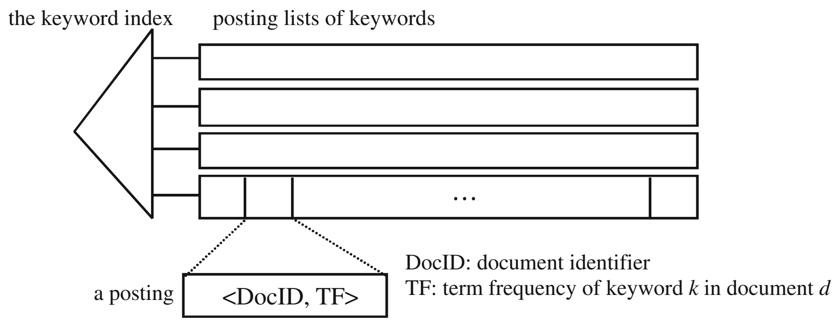

the keyword index
posting lists of keywords
...
a posting
<DocID, TF>
DocID: document identifier
TF: term frequency of keyword k in document d

Figure 2 The structure of the inverted index.

[35], several heuristic approaches [1, 6] have been proposed to find the top-k results; they maintain the postings in each posting list in the order of the term frequency (TF) or term frequency-inverse document frequency (TF-IDF) and compute the approximate scores of the objects accessing only part of the posting lists. Long et al. [20] propose a method that finds the top-k results using both PageRank and TF-IDF as ranking measures.

Guo et al. [13] propose an efficient method for updating the query-independent score of documents. The main idea is to divide each posting list into chunks, each of which is composed of postings with similar query-independent scores and a score range. The method reflects the updated score to the index only when the new score is out of the range of the chunk to which the document belongs. RASIM adopts a similar index structure in that objects are grouped based on the score. However, the work by Guo et al. cannot be applied to top-k spatial keyword queries due to the following distinct properties in top-k spatial keyword queries. First, the scores of the objects in top-k spatial keyword queries are determined dependent of the query while Guo et al. deal with the scores (such as PageRank) that can be determined independent of the query. Second, top-k spatial keyword queries deal with the scores that are calculated by component scores from multiple sources (i.e., text descriptions and location descriptions) while Guo et al. deal with the score that is from a single source. Thus, we need a new top-k pruning technique for dealing with the query-dependent scores from multiple sources.

Nearest neighbor queries A *nearest neighbor query* returns objects geographically closest to the query point. The incremental nearest neighbor (*Incremental NN*) algorithm proposed by Hjaltason et al. [17] is a representative work on nearest neighbor queries and is used by several existing works on top-k spatial keyword queries [10, 12]. Incremental NN finds objects closest to the query point one object at a time by computing the distance between the MBR of each node and the query point and by choosing the next node to access in the R-tree.

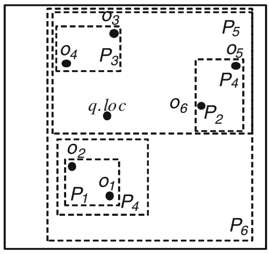
Spatial keyword queries Chen et al. [9] propose a separate index method that uses grid-based spatial structures in memory as the spatial index. This approach can efficiently merge the results from each index by clustering both spatial objects in each cell of the spatial index and the postings in each posting list in the same order of object IDs. The query processing algorithm merges the objects retrieved from the two indexes using the object IDs and filters out those merged objects that are not included in the query region by accessing the spatial objects themselves. This method handles merging efficiently, but cannot handle top-k pruning.

Zhou et al. [34] propose a method based on the hybrid index approach building the inverted index on each leaf node of the R*-tree, or the R*-tree on each posting list of the inverted index. Vaid et al. [28] presents an index whose structure is very similar to that of Zhou et al. [34]. Park et al. [23] propose a method that uses the R-tree as the spatial index and the S-tree, a hierarchical signature file with a structure symmetric to the R-tree, as the text index. This method connects each node of the R-tree to a node in the S-tree. Hariharan et al. [15] propose an R-tree-based method in which each node stores a list of keywords included in the objects of the node. Although all of these methods find the objects that satisfy spatial and keyword predicates, they do not handle top-k pruning.

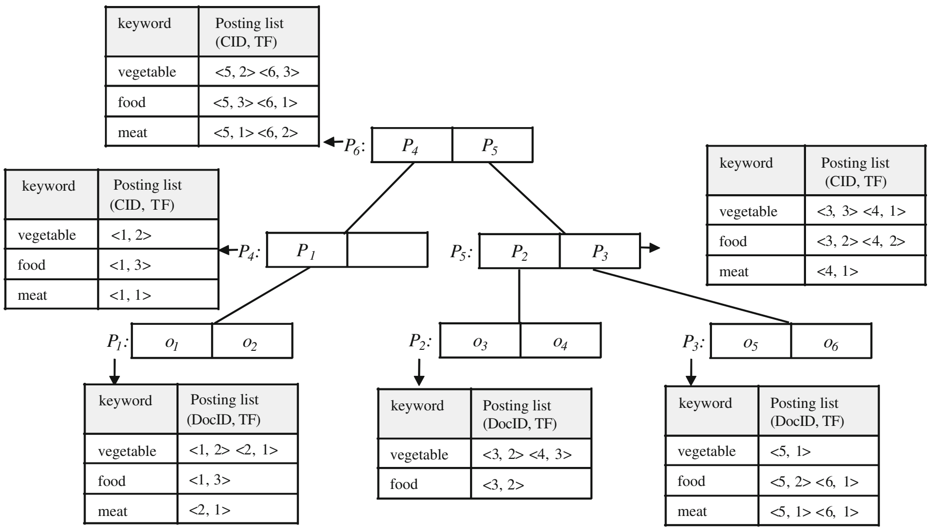
Top-k Spatial keyword queries Cong et al. [10] and Li et al. [19] propose the IR-tree method that supports top-k pruning based on the hybrid index approach. IR-tree builds the inverted index on each node of the R-tree. It extends Incremental NN [17] to use the score that combines spatial proximity and text relevancy instead of the distance. The score is derived from spatial proximity using MBRs in the R-tree and text relevancy using the inverted index. Figure 3 shows the structure of the IR-tree using the example in Figure 1. Figure 3a shows objects clustered by spatial proximity; Figure 3b shows the structure of the IR-tree. In each leaf node, the inverted index is built on the objects in the node. In each internal node, an inverted index is built on its children, and each posting is composed of the child ID (CID) for a child node and the maximum TF of the postings in the posting list of the corresponding keyword in the child node. We note that 1) the inverted indexes on internal nodes are redundant representation of the inverted indexes on the leaf nodes, which can be obviated in the separate index approach, and 2) the inverted indexes are stored in multiple files over different nodes.

Rocha-Junior et al. [24] propose the *Spatial Inverted Index* (simply, *S2I*). S2I is a method based on the hybrid index approach and builds the R-tree on each posting list of the inverted index. It retrieves the results by evaluating R-trees on the posting lists for the query keywords using Incremental NN. A major assumption in S2I is that only the objects including the query keywords can be the candidates for the result. However, typical top-k queries, including top-k spatial keyword queries, do not require this assumption. Due to this assumption, S2I cannot generate the same results as in the typical top-k spatial keyword queries. Figure 4 shows the structure of S2I for keywords k_1 and k_2. Let us suppose a query where $q.keywords = \{k_1\}$ to retrieve top-2 results according to the combined scores by summation of spatial proximity and text relevancy. Even if the correct answer is $\{o_5, o_1\}$, S2I retrieves $\{o_1, o_3\}$ as the results since it builds the index only for o_1 and o_3 and processes only them. This approximation affects the query performance since S2I considers only the objects including query keywords, but not all the objects based on the scoring function. Thus, it is unfair to directly compare RASIM with S2I since RASIM guarantees to provide correct results. The correctness of RASIM is proved in Theorems 1 and 2.

Martin et al. [21] propose a naive method based on the separate index approach. The method retrieves the results from each index, sorts them by each ranking measure, and then, computes combined scores by merging them. Meanwhile, Felipe et al. [12] deal with the problem using only spatial proximity as the ranking measure



(a) Clustering objects based on spatial proximity in the R-tree.



P_6:

keyword	Posting list (CID, TF)
vegetable	<5, 2> <6, 3>
food	<5, 3> <6, 1>
meat	<5, 1> <6, 2>

P_4:

keyword	Posting list (CID, TF)
vegetable	<1, 2>
food	<1, 3>
meat	<1, 1>

P_5:

keyword	Posting list (CID, TF)
vegetable	<3, 3> <4, 1>
food	<3, 2> <4, 2>
meat	<4, 1>

P_1:

keyword	Posting list (DocID, TF)
vegetable	<1, 2> <2, 1>
food	<1, 3>
meat	<2, 1>

P_2:

keyword	Posting list (DocID, TF)
vegetable	<3, 2> <4, 3>
food	<3, 2>

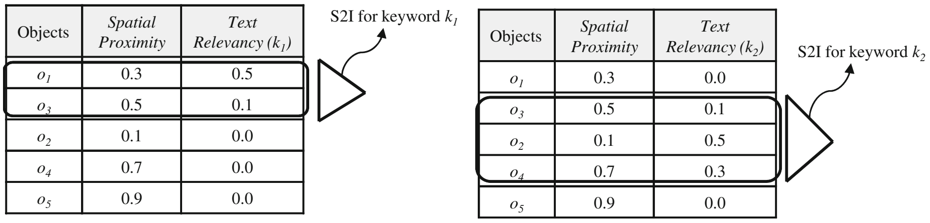
P_3:

keyword	Posting list (DocID, TF)
vegetable	<5, 1>
food	<5, 2> <6, 1>
meat	<5, 1> <6, 1>

(b) The structure of the IR-tree based on (a).

Figure 3 The structure of the IR-tree.

and keywords only as filtering conditions. In their method, they use an R-tree based index structure in which each node has additional signature information on text description of the objects in the node. This method cannot be applied to top-*k* spatial keyword queries, which consider not only spatial proximity but also text relevancy, since text relevancy cannot be effectively calculated using the signature information [35].


S2I for keyword k_1
S2I for keyword k_2

Objects	*Spatial Proximity*	*Text Relevancy (k_1)*
o_1	0.3	0.5
o_3	0.5	0.1
o_2	0.1	0.0
o_4	0.7	0.0
o_5	0.9	0.0

Objects	*Spatial Proximity*	*Text Relevancy (k_2)*
o_1	0.3	0.0
o_3	0.5	0.1
o_2	0.1	0.5
o_4	0.7	0.3
o_5	0.9	0.0

Figure 4 The structure of S2I.

3 Problem definition

In this section, we formally define the problem of the top-k spatial keyword query. Let D be a database. Each object o in D is defined as a pair $(o.loc, o.doc)$ where $o.loc$ is the location description in a multidimensional space and $o.doc$ is the text description that describes the object. We represent $o.loc$ as an MBR as in IR-tree [10]. Table 1 summarizes the notation. A top-k spatial keyword query q is defined as a 4-tuple $(q.loc, q.keywords, q.k, q.p)$. Top-$k$ spatial keyword queries retrieve k objects with the highest (or lowest) combined scores. Without loss of generality, in the rest of this paper, we assume that we are looking for objects having the highest scores.

We use the following combined scoring function $f(q, o)$ as defined in [10].

$$f(q,o) = q.p * f_{SP}(q.loc, o.loc)/maxSP + (1 - q.p) * f_{TR}(q.keywords, o.doc)/maxTR \tag{1}$$

The *component scores* $f_{SP}(q.loc, o.loc)$ and $f_{TR}(q.keywords, o.doc)$ are normalized by $maxSP$ and $maxTR$, respectively. In (1), $f(q, o)$ uses the OR semantics [9] to combine $f_{SP}()$ and $f_{TR}()$. Hence, objects that do not include $q.keywords$ (i.e., $f_{TR}() = 0$) may still belong to the result sets according to their combined scores.

The value of $f_{SP}(q.loc, o.loc)$ can be computed according to any measure that reflects the spatial similarity between an object and a query point. In this paper, we use the Euclidian distance ED, and $f_{SP}(q.loc, o.loc)$ is computed by $(maxSP - ED(q.loc, o.loc))$. The higher the value of $f_{SP}(q.loc, o.loc)$, the more relevant it is to the query.

The value of $f_{TR}(q.keywords, o.doc)$ can be computed according to any measure that reflects the similarity between a set of keywords and an object. In

Table 1 The notation.

Symbols	Definitions
$q.loc$	The location description specified in q
$q.keywords$	The set of keywords $\{k_1, k_2, ..., k_n\}$ specified in q
$q.k$	The desired number of results for q
$q.p$	The user preference: the weight for the spatial component score (1-$q.p$: the weight for the keyword component score)
$f(q, o)$	The combined score of object o with respect to the query q
$f_{SP}(loc_i, loc_j)$	Spatial component score computed from spatial proximity between location descriptions loc_i and loc_j
$f_{TR}(q.keywords, o.doc)$	Keyword component score computed from text relevancy between $q.keywords$ and text description $o.doc$ of the object o
$TF(o.doc, k_i)$	Term frequency for the keyword k_i in the text description $o.doc$ of the object o
$DF(D, k_i)$	Document frequency for the keyword k_i in D (i.e., the number of postings in the posting list for the keyword k_i)
N	The number of objects in the database
$ED(loc_i, loc_j)$	The Euclidean distance between location descriptions loc_i and loc_j
$maxSP$	The maximum spatial component score: $max_{o_x, o_y \in D}\ ED(o_x.loc, o_y.loc)$
$maxTR$	The maximum keyword component score: $max_{o \in D}\ f_{TR}(q.keywords, o.doc)$

this paper, we use the *term-weighting scheme* [3] as shown in (2), which is the best known query-dependent score for measuring text relevancy. The higher the value of $f_{TR}(q.keywords, o.doc)$, the more relevant it is to the query. In $f_{TR}(q.keywords, o.doc)$ we assume the OR semantics [20]; i.e., objects including only a subset of $q.keywords$ can have scores as well.

$$f_{TR}(q.keywords, o.doc) = \sum_{i=1}^{n} f_{TR}(q.k_i, o.doc) = \sum_{i=1}^{n} TF(o.doc, k_i)\ log \frac{N}{DF(D, k_i)} \tag{2}$$

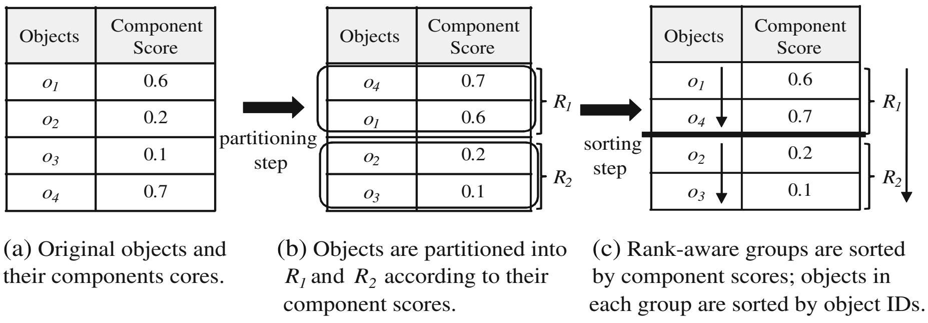
In this paper, we focus on the efficiency of query processing and not on the effectiveness of a ranking function. In addition to (1), any arbitrary ranking function that is monotone with respect to the spatial component score and the keyword component score can be used.

4 The rank-aware separate index method (RASIM)

4.1 The concept

Top-k spatial keyword query processing based on the separate index approach requires two different orders for clustering: (i) ordering for efficient merging of objects retrieved from the spatial index and from the text index and (ii) ordering for top-k pruning. For the former, we can order objects according to their object IDs. For the latter, we can order objects according to their component scores and find the top-k results by applying top-k query processing methods such as Threshold Algorithm [11]. We note that we should consider two different orders for clustering simultaneously for top-k spatial keyword queries.

In this section, we present the basic idea of RASIM. RASIM supports clustering objects by two different orders through a two-step process: the partitioning step and the sorting step as shown in Figure 5. The former partitions the set of objects into groups that contains objects with similar component scores. We call each group of objects in Figure 5 a *rank-aware group*. The latter sorts (i) rank-aware groups by scores and (ii) objects in each rank-aware group by object IDs.



Objects
Component Score
o_1 0.6
o_2 0.2
o_3 0.1
o_4 0.7
partitioning step
o_4 0.7
o_1 0.6
o_2 0.2
o_3 0.1
R_1
R_2
sorting step
o_1 0.6
o_4 0.7
o_2 0.2
o_3 0.1
(a) Original objects and their components cores.
(b) Objects are partitioned into R_1 and R_2 according to their component scores.
(c) Rank-aware groups are sorted by component scores; objects in each group are sorted by object IDs.

Figure 5 The two-step process for clustering objects by two different orders.

For the rest of the paper, we denote "rank-aware" as *RA*. Each RA group R has a range of component scores bounded by *lower bound*(R), which is a score lb satisfying that $lb <= component\ score(o)$ for all o in R, and upper bound(R), which is a score ub satisfying that $ub >= component\ score(o)$ for all o in R. We will explain how to determine *lower bound*(R) and *upper bound*(R) for each component score in Section 4.3.1. We note that the component scores of objects for a top-k spatial keyword query can be computed only when the query is given. Thus, we partition the set of objects according to their *similarity* such as spatial proximity or text relevancy used for calculating the component scores rather than component scores themselves; we then obtain a sorted list of the groups according to component scores dependent of the query when the query is given.

Based on the concept of the RA group, we propose two query processing algorithms: *External TA* and *Generalized External TA*. External TA extends Threshold Algorithm (TA) [11], which uses individual objects as the unit, to use the RA groups as the unit. Generalized External TA enhances External TA. In External TA, we observe two types of inefficiencies: (i) it accesses two lists equally regardless of the user preference even if it is desirable to process more objects from the list with a higher weight and (ii) unnecessary object accesses are required due to the overlap of the score ranges among the RA groups. Generalized External TA solves these types of inefficiencies of External TA by generalizing the concept of the RA group. We elaborate on these algorithms in Section 4.3.

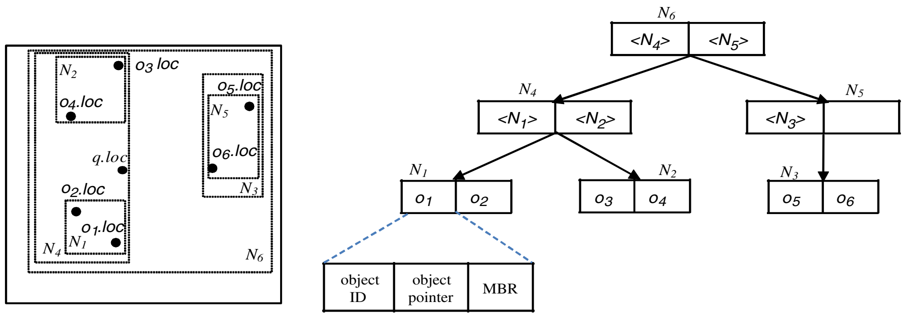
4.2 RA separate indexes

RASIM uses an *RA spatial index* for search by location and an *RA inverted index* for search by text. Collectively, we call them *RA separate indexes*. Each index consists of a set of RA groups; objects[1] in each group are sorted by their object IDs. We use the RA group as the unit of physical access mapped to one disk page.

The RA spatial index The *RA spatial index* is an index for spatial data that consists of a multilevel directory and leaf nodes. In the RA spatial index, an entry in the leaf-level node represents the MBR and the pointer to an object; an entry in a non-leaf-level node represents the MBR containing the objects in a child directory node or a leaf node together with the pointer to the node. Each leaf node of the RA spatial index contains a set of objects clustered by spatial proximity. The RA spatial index regards each leaf node as an RA group; entries in each group are sorted according to their object IDs.[2] Each entry in the group is a triple <object ID, object pointer, MBR>. Options for the RA spatial index include the R-tree family [4, 14, 25] and the MBR-MLGF [26].

[1] Objects are actually stored in the data file, and the index has an entry for each object that indicates the information on the object such as the object ID and the pointer to that object. For convenience, however, when there is no ambiguity, we denote an entry as an object.

[2] This step can be omitted if the objects are already in the object ID order. For example, when MBR-MLGF [26] is used for the RA spatial index, it maintains the objects in each leaf node according to the Z-order. In that case, we do not need to sort the objects in the RA group if we use the Z-order values themselves as object IDs. If the index requires a different order among the entries in a leaf node, the index access algorithm may have to be slightly modified to accommodate the order RASIM requires.



(a) Representation of objects and MBRs.

(b) The structure of the RA spatial index for (a). $<N_i>$ represents the pointer to the node N_i. Leaf nodes N_1,N_2,N_3 are RA groups.

Figure 6 An example RA spatial index.

Example 1 Figure 6a shows objects and MBRs of objects in an RA spatial index. Figure 6b shows the structure of the RA spatial index for the objects shown in Figure 6a. N_4, N_5, and N_6 are non-leaf pages; N_1, N_2, and N_3 leaf pages, each of which is an RA group.

The RA inverted index In the RA inverted index, each keyword corresponds to a posting list. Each posting list is partitioned into RA groups according to text relevancy. As shown in (2), text relevancy of the postings in a posting list for keyword k_i can be determined by $TF(o_j.doc, k_i)$ since all the postings in the posting list have the same $DF(D, k_i)$ and N. RA groups in each posting list are maintained in the order of $TF(o_j.doc, k_i)$; postings in each group are sorted according to their object IDs. Each posting is a triple $<$object ID, object pointer, $TF(o_j.doc, k_i)>$. The RA inverted index uses the subindex [31, 32] to support search for the specific posting of an object.

Example 2 Figure 7 shows an example RA inverted index. $R_{vegetable,1}$ indicates the RA group that contains the objects with the highest TFs in the posting list of the keyword 'vegetable'.

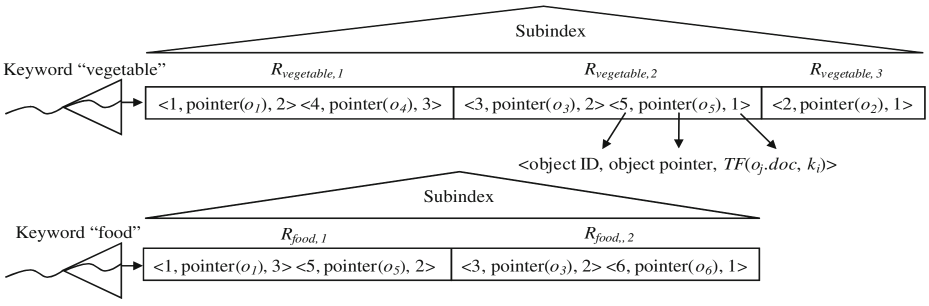

Subindex
Keyword "vegetable"
R vegetable, 1
R vegetable, 2
R vegetable, 3
<1, pointer(o1), 2> <4, pointer(o4), 3>
<3, pointer(o3), 2> <5, pointer(o5), 1>
<2, pointer(o2), 1>
<object ID, object pointer, TF(oj.doc, ki)>
Subindex
Keyword "food"
R food, 1
R food, 2
<1, pointer(o1), 3> <5, pointer(o5), 2>
<3, pointer(o3), 2> <6, pointer(o6), 1>

Figure 7 An example RA inverted index.

4.3 Query processing algorithms based on the RA separate indexes

We first present in Section 4.3.1 a method to obtain a sorted list of the RA groups according to their component scores from the RA separate indexes when the query is given. We then present two query processing algorithms—External TA and Generalized External TA—in Sections 4.3.2 and 4.3.3, respectively.

4.3.1 Dynamic sorted RA group list

Definition 1 The *sorted RA group list* L is defined as the list of RA groups $\{R_i\}$ $(1 \leq i \leq n)$ that are sorted by upper bound(R_i). The *current RA group* of L is defined as the RA group that is being processed for the query. The *next RA group* of L is defined as the RA group R_{i+1} to access after the current RA group R_i in L.

We denote the sorted RA group list for the spatial component score as L_{SP} and that for the keyword component score as L_{TR}. We define $\overline{S_{SP}}$ $(\overline{S_{TR}})$ as an upper bound of the next RA group in L_{SP} (L_{TR}). We construct the sorted RA group lists L_{SP} and L_{TR} by accessing the current RA group one by one from the RA spatial index and the RA inverted index, respectively.

The sorted RA group list L_{SP} To obtain the current RA group of L_{SP}, we apply the Incremental NN algorithm [17] in the unit of RA groups instead of individual objects. For Incremental NN, we maintain a priority queue that sorts the RA groups by their upper bounds. We obtain the current RA group from the front of the priority queue.[3] Here, we use $f_{SP}(q.loc, \text{MBR}(R))$ as upper bound (R) of an RA group R since the spatial component score $f_{SP}()$ of any object in R cannot be higher than $f_{SP}(q.loc, \text{MBR}(R))$. We can compute $f_{SP}(q.loc, \text{MBR}(R))$ without accessing R, which is a leaf node of the RA spatial index, since MBR(R) can be obtained from the parent of R in the RA spatial index. $\overline{S_{SP}}$ can be obtained as $f_{SP}(q.loc, \text{MBR}(NG))$ where NG is the next RA group of L_{SP} and can be obtained from the priority queue.

Example 3 Let us consider $q.loc$ in Figure 1 as $q.loc$ of a top-k spatial keyword query. In Figure 8, we obtain the spatial component scores of objects and RA groups with respect to $q.loc$ from Figure 6a. Here, the spatial component score $f_{SP}()$ is calculated as $(maxSP - ED(q.loc,o.loc))$ as explained in Section 3. The current RA group of L_{SP} is $N_1 = \{(o_1, 0.6), (o_2, 0.7)\}$ since upper bound (N_1), i.e., $f_{SP}(q.loc, \text{MBR}(N_1))$, is the highest among those of the RA groups. $\overline{S_{SP}} = 0.5$ from $f_{SP}(q.loc, \text{MBR}(N_2))$ since N_2 is the next RA group of L_{SP}.

The sorted RA group list L_{TR} When a single query keyword is given, the current RA group is obtained from the posting list for the keyword. When multiple query keywords are given, the current RA group is obtained by merging the current RA groups from multiple posting lists—one for each query keyword. We first summarize the notion for the sorted RA group list L_{TR} in Table 2.

[3]When we use the R+-Tree [4] as the RA spatial index, we need to eliminate duplicated objects since an object may be stored in multiple leaf nodes.

Figure 8 The spatial component scores $f_{SP}()$ of objects and RA groups with respect to $q.loc$.

	Objects	$f_{SP}(q.loc, o_x)$
N_1	o_1	0.6
	o_2	0.7
N_2	o_3	0.3
	o_4	0.4
N_3	o_5	0.2
	o_6	0.4

RA groups	$f_{SP}(q.loc, \mathrm{MBR}(N_y))$
N_1	0.8
N_2	0.5
N_3	0.4

Figure 9 shows the algorithm for retrieving the current RA group of L_{TR}. The input to the algorithm is $q.keywords$. We obtain the current RA group from $R_{candidates}$, which is obtained by n-way merging the current RA group (k_i) $(k_i \in q.keywords)$ according to object IDs. Since the current RA group (k_i) $(k_i \in q.keywords)$ is mapped to a disk page, we can access the objects with similar keyword component scores for each keyword by a disk I/O. When there are multiple keywords in $q.keywords$, since the size of $R_{candidates}$ could be different from that of an RA group, N_P, we retrieve N_P objects as a unit. We retrieve N_P objects of $R_{candidates}$ as the current RA group in the order of the object ID; we maintain the remaining objects in $R_{candidates}$ for the next RA group. If the current size of $R_{candidates}$ is less than N_P, we access the next RA group (k_i) $(k_i \in q.keywords)$ and merge them into $R_{candidates}$ until the size of $R_{candidates}$ is larger than or equal to N_P except when no more RA group for k_i remains. $f_{TR}(q.keywords, o.doc)$ of an object o in the current RA group is computed as $\sum_{i=1}^{n} f_{TR}(k_i, o.doc)$ by (2). If o is not included in the current RA group (k_i) for keyword k_i, we obtain $f_{TR}(k_i, o.doc)$ by searching for a posting of o from the posting list for k_i using the subindex.

$\overline{S_{TR}}$ is determined as the maximum value of (i) the highest value of keyword component scores $f_{TR}(q.keywords, o.doc)$ of the objects remaining in $R_{candidates}$ and (ii) an upper bound of keyword component scores of the objects that have not yet been included in $R_{candidates}$. The latter can be obtained as $\sum_{i=1}^{n} \overline{S_{TR}}(k_i)$ since each $\overline{S_{TR}}(k_i)$ is an upper bound of the next RA group (R_i). Here, we use the lowest value of $f_{TR}(k_i, o.doc)$ of the objects in the current RA group (k_i) as $\overline{S_{TR}}(k_i)$ since RA groups R_j $(1 \le j \le n)$ in $L_{TR}(k_i)$ are disjoint.

Table 2 Summary of notation for L_{TR}.

Symbols	Definitions
N_P	The maximum number of postings stored in an RA group
$L_{TR}(k_i)$	The RA groups R_i $(1 \le i \le n)$ that are sorted by TFs in the posting list for keyword k_i
The current RA group (k_i)	The RA group that is being processed in $L_{TR}(k_i)$
The next RA group (k_i)	The RA group R_{i+1} to access after the current RA group (k_i) R_i in $L_{TR}(k_i)$
$\overline{S_{TR}}(k_i)$	An upper bound of the next RA group (k_i)
$R_{candidates}$	The objects that have so far been included in the current RA group (k_i) $(k_i \in q.keywords)$ but that have not yet been included in the current RA group of L_{TR}

```
Algorithm $RetrieveCurrentRAGroupOfL_{TR}$:
1:  Input: $q.keywords$: $\{k_1, k_2, \ldots, k_n\}$
2:  Output: $CG_{TR}$: the current RA group of $L_{TR}$
3:  $CG_{TR}$ := {};
4:  $R_{temp}$ :={};
     /* $R_{candidates}$ stores the objects that have so far been included in the current RA group($k_i$)
        ($k_i \in q.keywords$ ) but that have not yet been included in the current RA group of $L_{TR}$ */
5:  IF $|R_{candidates}| \geq N_P$ THEN BEGIN
6:      Store $N_P$ objects of $R_{candidates}$ into $CG_{TR}$ in the order of object ID;
7:      $R_{candidates} := R_{candidates} - CG_{TR}$;
8:      END
9:  ELSE BEGIN
10:     WHILE $|R_{candidates}| < N_P$ DO BEGIN
            /* $R(k_i)$ denotes the current RA group ($k_i$) */
11:         Store a list of objects obtained by $n$-way merging (i.e., outer join) of $R(k_i)$'s
            ($k_i \in q.keywords$ ) into $Merged$;
12:         WHILE objects are remaining in $Merged$ DO BEGIN
13:             $o$ := the first element in $Merged$;
14:             IF $o$ has not yet been included in $R_{candidates}$ THEN BEGIN
15:                 $f_{TR}(q.keywords, o.doc)$ := 0;          /* initialize */
16:                 FOR $i$ := 1 to $n$ DO BEGIN           /* for each keyword $k_i$ in $q.keywords$ */
17:                     IF $o$ is included in the current RA group($k_i$) THEN
18:                         Compute $f_{TR}(k_i, o.doc)$;
19:                     ELSE BEGIN
20:                         Search for the posting of $o$ from the posting list for $k_i$ using subindex;
21:                         Compute $f_{TR}(k_i, o.doc)$;
22:                         END
23:                     $f_{TR}(q.keywords, o.doc) := f_{TR}(q.keywords, o.doc) + f_{TR}(k_i, o.doc)$;
24:                     END
25:                 IF $|R_{candidates}| < N_P$ THEN
26:                     Merge <$o, f_{TR}(q.keywords, o.doc)$> into $R_{candidates}$ in the order of object IDs;
27:                 ELSE
28:                     Append <$o, f_{TR}(q.keywords, o.doc)$> into $R_{temp}$;
29:                 Mark that $o$ has been included in $R_{candidates}$ in the hash table;
30:                 END
31:             Advance $o$ to the next object in $Merged$;
32:             END
33:         FOR $i$ := 1 to $n$ DO                  /* for each keyword $k_i$ in $q.keywords$ */
34:             Advance $R(k_i)$ to the next RA group($k_i$);
35:         END
36:     $CG_{TR} := R_{candidates}$;
37:     $R_{candidates} := R_{temp}$;
38:     END
39: RETURN $CG_{TR}$;
```

Figure 9 The algorithm for retrieving the current RA group of L_{TR}.

Example 4 Let us consider {‘vegetable, ‘food’} as *q.keywords* of a top-*k* spatial keyword query. In Figure 10, we obtain the keyword component scores of the objects with respect to *q.keywords* from Figure 7. Here, N_P (i.e., the maximum number of postings stored in an RA group) is 2. The keyword component score $f_{TR}()$ is calculated using (2). For example, $f_{TR}(q.keywords, o_1) = f_{TR}(\text{‘vegetable’}, o_1) + f_{TR}(\text{‘food’}, o_1)$ by (2); $f_{TR}(\text{‘vegetable’}, o_1) = 0.158$ since $TF(o_1, \text{‘vegetable’}) = 2$, $N = 6$, and $DF(D, \text{‘vegetable’}) = 5$; $f_{TR}(\text{‘food’}, o_1) = 0.528$ since $TF(o_1, \text{‘food’}) = 3$ and $DF(D, \text{‘food’}) = 4$.

The current RA group of L_{TR} is obtained from $R_{candidates}$; $R_{candidates}$ is obtained by merging the current RA group (‘vegetable’) and the current RA group (‘food’). Hence, $R_{candidates} = \{(o_1, 0.686), (o_4, 0.237), (o_5, 0.431)\}$; the current RA group of L_{TR} is $\{(o_1, 0.686), (o_4, 0.237)\}$; o_5 remains in $R_{candidates}$ for the next RA group. $\overline{S_{TR}} = 0.510$ from the maximum value of (i) $f_{TR}(q.loc, o_5) = 0.431$ (i.e., the highest keyword component score $f_{TR}()$ among the objects remaining in $R_{candidates}$) and (ii) $\sum_{i=1}^{n} \overline{S_{TR}}(k_i) = 0.510$ (i.e., $\overline{S_{TR}}(\text{‘vegetable’}) = 0.158$ and $\overline{S_{TR}}(\text{‘food’}) = 0.352$).

4.3.2 External Threshold Algorithm (External TA)

The Threshold Algorithm (TA) [11] is a method that supports top-*k* pruning by using individual objects as the unit when sorted lists are given. TA retrieves *k* objects with the highest combined scores while accessing sorted lists in parallel until top-*k* objects whose scores are higher than the threshold value are obtained. TA maintains the threshold value by applying the component score of the last object accessed from each list into the scoring function. If an object *o* is accessed from only one list, TA randomly accesses *o* in the other list to obtain the component score.

To support top-*k* pruning when using the RA group as the unit, we propose *External TA* that extends TA to handle the RA group rather than individual objects. We use the RA group as the unit of I/O by mapping it to one disk page. We note that the objects in an RA group have similar component scores and are retrieved by one disk I/O operation. To explain External TA, we first define the *threshold value* in Definition 3.

Definition 2 The *threshold value* is defined as (1) obtained by substituting the spatial component score $f_{SP}()$ with $\overline{S_{SP}}$ and the keyword component score $f_{TR}()$ with $\overline{S_{TR}}$.

Figure 11 shows External TA. The inputs are the lists L_{SP}, L_{TR}, and a top-*k* spatial keyword query *q*; the output is the list of top-*k* objects with the highest combined

	Objects	f_{TR}(‘vegetable’, o_x)
$R_{vegetable,\,1}$	o_1	0.158
	o_4	0.237
$R_{vegetable,\,2}$	o_3	0.158
	o_5	0.079
$R_{vegetable,\,3}$	o_2	0.079

	Objects	f_{TR}(‘food’, o_x)
$R_{food,\,1}$	o_1	0.528
	o_5	0.352
$R_{food,\,2}$	o_3	0.352
	o_6	0.176

Objects	$f_{TR}(q.keywords, o_x)$
o_1	0.686
o_2	0.079
o_3	0.510
o_4	0.237
o_5	0.431
o_6	0.176

Figure 10 The keyword component score $f_{TR}()$ of objects with respect to *q.keywords*.

```
Algorithm ExternalTA:
1:  Input: (1) $L_{SP}$, (2) $L_{TR}$, (3) $q$: a top-$k$ spatial keyword query
2:  Output: Result: the list of top-$k$ objects with the highest combined scores $f()$
3:  CSLowerThanThreshold:= {};            /* initialize CSLowerThanThreshold */
4:  CSHigherThanThreshold :={};           /* initialize CSHigherThanThreshold */
5:   REPEAT
     /* Step 1: Retrieve the current RA groups and compute the threshold value Threshold */
6:   Retrieve the current RA groups $CG_{SP}$ of $L_{SP}$ and $CG_{TR}$ of $L_{TR}$;
7:   Compute Threshold by plugging $\overline{S_{SP}}$ and $\overline{S_{TR}}$ into Eq. (1) as the component scores;
     /* Step 2: Examine the objects in CSLowerThanThreshold */
8:   FOR each $o \in$ CSLowerThanThreshold DO BEGIN
9:      IF $f(q, o) >$ Threshold  THEN BEGIN
10:           CSHigherThanThreshold := CSHigherThanThreshold $\cup \{o\}$;
11:           CSLowerThanThreshold:= CSLowerThanThreshold $- \{o\}$;
12:           END
13:      END
      /* Step 3: Examine the objects in the current groups */
      /* $CG_{SP}$ and $CG_{TR}$ are in the order of the object ID */
14:   $i$ := the first element in $CG_{SP}$;  $j$ := the first element in $CG_{TR}$;
15:   WHILE objects are remaining in either $CG_{SP}$ or $CG_{TR}$ DO BEGIN
          /* Step 3-1: Compute the combined score of each object $o$ */
16:       IF object ID($i$) = object ID($j$) THEN BEGIN
17:            $o$ := $i$;                 /* $o$ is retrieved from both lists $L_{SP}$ and $L_{TR}$ */
18:            Compute the combined score $f(q, o)$ in Eq. (1);
19:            Advance $i$ and $j$ to the next object in $CG_{SP}$ and $CG_{TR}$, respectively;
20:            END
21:       ELSE IF object ID($i$) > object ID($j$) THEN BEGIN
22:            $o$ := $j$;                 /* $o$ is retrieved only from $L_{TR}$ */
23:            Do random access to the other list $L_{SP}$ to obtain the spatial component score of $o$;
24:            Compute the combined score $f(q, o)$ in Eq. (1);
25:            Advance $j$ to the next object in $CG_{TR}$;
26:            END
27:       ELSE  BEGIN /* object ID($i$) < object ID($j$)  */
28:            $o$ := $i$;                 /* $o$ is retrieved only from $L_{SP}$ */
29:            Do random access to the other list $L_{TR}$ to obtain keyword the component score of $o$;
30:            Compute the combined score $f(q, o)$ in Eq. (1);
31:            Advance $i$ to the next object in $CG_{SP}$;
32:            END
          /* Step 3-2: Store the object  according to its combined score */
33:       IF $f(q, o) >$ Threshold THEN
34:            CSHigherThanThreshold := CSHigherThanThreshold $\cup \{o\}$;
35:       ELSE /*  $f(q, o) \leq$ Threshold  */
36:            CSLowerThanThreshold:= CSLowerThanThreshold $\cup \{o\}$;
37:       END
38:   UNTIL |CSHigherThanThreshold | $\geq k$
      /* Step 4: Retrieve the top-$k$ results */
39:   RETURN top-$k$ objects in CSHigherThanThreshold ;
```

Figure 11 External TA.

scores in (1). The algorithm initializes *CSLowerThanThreshold*, the objects whose combined scores (CS) are lower than or equal to the threshold value, and *CSHigherThanThreshold*, the objects whose combined scores are higher than the threshold value. These objects are maintained in the order of the combined score in a priority queue. The algorithm iteratively performs the following three steps until the number of the objects in *CSHigherThanThreshold* becomes higher than or equal to k. In Step 1, the algorithm first retrieves the current RA groups of L_{SP} and L_{TR}. It then computes the threshold value by plugging $\overline{S_{SP}}$ and $\overline{S_{TR}}$ into (1) as the component scores. In Step 2, the algorithm examines the objects in *CSLowerThanThreshold*. If the combined score of an object in *CSLowerThanThreshold* is higher than the threshold value, it stores the object in *CSHigherThanThreshold*. In Step 3, the algorithm examines the objects in the current RA groups. First, it computes the combined scores of the objects in the current RA groups. If an object o is retrieved from both lists L_{SP} and L_{TR}, the algorithm directly computes the combined score; if o is retrieved from only one list L_i, the algorithm randomly accesses the other list L_j ($i \neq j$) to obtain the component score of o, and then, computes the combined score $f(q, o)$. Randomly accessing object o can be supported differently for each list. We can easily obtain the spatial component score by accessing o in the data file through a pointer to o maintained in the index since the location description for computing the spatial component score is stored in the data file. However, we cannot obtain the keyword component score from the data file since $TF(k_i, o.doc)$ and $DF(k_i, D)$ for computing the keyword component score are not stored in the data file, but in the index. In that case, we obtain it by searching for the posting of o from each posting list for *q.keywords* using the subindex. Next, if the combined score $f(q, o)$ is higher than the threshold value, it stores the object in *CSHigherThanThreshold*; if not, it stores the object in *CSLowerThanThreshold*. In Step 4, the algorithm returns top-k objects in *CSHigherThanThreshold*.

Theorem 1 *External TA correctly finds the top-k results.*

Proof In External TA, scores used for computing the threshold value are different from those in TA. Hence, while TA uses the component score of the last object accessed from each list, External TA uses $\overline{S_{SP}}$ and $\overline{S_{TR}}$. We need to prove that $\overline{S_{SP}}$ ($\overline{S_{TR}}$) is an upper bound of component scores of the objects that have not yet been retrieved from the list L_{SP} (L_{TR}). This holds for the following reasons: (i) $\overline{S_{SP}}$ ($\overline{S_{TR}}$) is obtained from the next RA group R whose upper bound (R) is the highest among those of the RA groups that have not yet been retrieved from the list L_{SP} (L_{TR}) and (ii) $\overline{S_{SP}}$ ($\overline{S_{TR}}$) is an upper bound of the component scores of the objects in the next RA group of L_{SP} (L_{TR}). The rest of the proof is the same as that for TA in Fagin et al. [11]. □

Example 5 Let us consider a top-k spatial keyword query q (*q.loc*, {'vegetable', 'food'}, 1, 0.5) for the database in Figure 1. Thus, *q.keywords* = {'vegetable', 'food'}, $q.k = 1$, and $q.p = 0.5$. We assume $maxSP$ and $maxTR$ are 1.0. The steps for processing the query in External TA are as follows.

1. Retrieve the current RA groups:
   - The current RA group of L_{SP} is $\{(o_1, 0.6), (o_2, 0.7)\}$ (from Example 3)

  - The current RA group of L_{TR} is {(o_1, 0.686), (o_4, 0.237)} (from Example 4)

2. Compute the threshold value:
  - $\overline{S_{SP}} = 0.5$ and $\overline{S_{TR}} = 0.51$ (from Examples 3 and 4)
  - Threshold value = 0.505 (from (1) with $\overline{S_{SP}} = 0.5$, $\overline{S_{TR}} = 0.510$)

3. Calculate the combined scores of the objects in the current RA groups:
  - $f(q, o_1) = 0.643$ (from (1) with $f_{SP}(q.loc, o_1) = 0.6$ and $f_{TR}(q.loc, o_1) = 0.686$)
  - $f(q, o_2) = 0.3895$ (from (1) with $f_{SP}(q.loc, o_2) = 0.7$ and $f_{TR}(q.loc, o_2) = 0.079$)
  - $f(q, o_4) = 0.3185$ (from (1) with $f_{SP}(q.loc, o_4) = 0.4$ and $f_{TR}(q.loc, o_4) = 0.237$)

o_1 is stored in *CSHigherThanThresholdValue* and is returned as the output since $f(q, o_1) >$ threshold value.

4.3.3 Generalized External TA

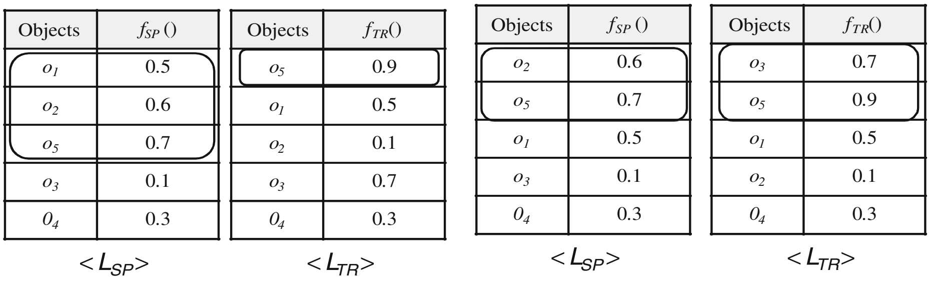
In this section, we generalize the notion of the RA group as follows: (i) the RA group has an arbitrary size mapped to a set of pages and (ii) score ranges of RA groups may overlap. Then, we present *Generalized External TA* that enhances the performance of External TA.

We first generalize the RA group such that an RA group has an arbitrary size. The group size is the number of pages (or the number of objects) in an RA group. We regard the original TA that deals with individual objects as a special case where each group consists of one object. By controlling the group size, RASIM can be made a method handling only top-*k* pruning or a method handling only efficient merging at extremes. That is, if the group size is one object, RASIM becomes the former. If the group size $\geq x$ pages, where x pages contain all the objects that can be the results, RASIM becomes the latter. Since a large group size means that many objects can be accessed and processed at a time, a larger group size means better efficiency for large k.

Based on this generalization, we propose a *weighted access technique* that controls the group size for each list in proportion to the user preference. By using a larger group size for a list with a higher weight, we can enhance the performance since (i) the threshold value can be decreased fast as we proceed and (ii) more objects can be processed from the list with a higher weight. We present an intuitive example in Example 6.

Example 6 Figure 12a shows the process of accessing the lists L_{SP} and L_{TR} in proportion to the user preference; Figure 12b shows the process of accessing the lists L_{SP} and L_{TR} with an equal proportion regardless of the user preference. Here, we assume $q.k = 3$, $q.p = 0.75$, $maxSP = 1.0$, and $maxTR = 1.0$. In Figure 12a, after accessing the current RA groups for each list, the threshold value and the combined scores of the objects are as follows:

- Threshold value = 0.4 (from (1) with $\overline{S_{SP}} = 0.3$, $\overline{S_{TR}} = 0.7$)
- $f(q, o_1) = 0.5$; $f(q, o_2) = 0.475$; $f(q, o_5) = 0.75$ (from (1) with $f_{SP}()$ and $f_{TR}()$ in Figure 12)



Objects	$f_{SP}()$
o_1	0.5
o_2	0.6
o_5	0.7
o_3	0.1
o_4	0.3

$<L_{SP}>$

Objects	$f_{TR}()$
o_5	0.9
o_1	0.5
o_2	0.1
o_3	0.7
o_4	0.3

$<L_{TR}>$

(a) Accessing the lists in proportion to the user preference.

Objects	$f_{SP}()$
o_2	0.6
o_5	0.7
o_1	0.5
o_3	0.1
o_4	0.3

$<L_{SP}>$

Objects	$f_{TR}()$
o_3	0.7
o_5	0.9
o_1	0.5
o_2	0.1
o_4	0.3

$<L_{TR}>$

(b) Accessing the lists with an equal proportion.

Figure 12 An example for weighted access ($q.p = 0.75$).

In Figure 12(b), after accessing the current RA groups for each list, the threshold value and the combined scores of the objects are as follows:

- Threshold value $= 0.5$ (from (1) with $\overline{S_{SP}} = 0.5$, $\overline{S_{TR}} = 0.5$)
- $f(q, o_2) = 0.475$; $f(q, o_3) = 0.25$; $f(q, o_5) = 0.75$ (from (1) with $f_{SP}()$ and $f_{TR}()$ in Figure 12)

We observe that we can retrieve all top-3 results in Figure 12a since the combined scores of o_1, o_2, and o_5 are higher than the threshold value. However, we retrieve only o_5 in Figure 12b.

Next, we generalize the RA group such that component score ranges of the RA groups may overlap. By overlapping we mean the lowest component score of objects in the current RA group may be lower than the upper bound of the objects that have not yet been processed. The reason of this overlap in L_{SP} is as follows. The order of RA groups in L_{SP} is determined by $f_{SP}(q.loc, \text{MBR}(R))$, where R is an RA group. $f_{SP}(q.loc, \text{MBR}(CG))$, where CG is the current RA group, is the highest among those of RA groups in L_{SP}. However, $f_{SP}(q.loc, \text{MBR}(CG))$ is an upper bound of the spatial component scores of all the objects in CG. Thus, it does not guarantee that the component score of every object in CG is higher than those of the objects in the next RA group. This lack of guarantee comes from the fact that each RA group is constructed not by the spatial component scores of objects but by similarity (i.e., spatial proximity) of objects. For example, in Figure 6a, while upper bound (N_2), $f_{SP}(q.loc, \text{MBR}(N_2))$, is higher than, upper bound (N_3), $f_{SP}(q.loc, \text{MBR}(N_3))$—i.e., MBR($N_2$) is closer to q than MBR(N_3)—the spatial component score $f_{SP}(q.loc, o_3)$ of o_3 in N_2 is lower than the spatial component score $f_{SP}(q.loc, o_6)$ of o_6 in N_3—i.e., o_3 is farther from q than o_6 (see Figure 8).

The reason that the component score ranges of RA groups in L_{TR} may overlap is as follows. The component score ranges of the RA groups for each keyword are partitioned. However, if we merge the current RA group (k_i) for all $k_i \in q.keywords$ to obtain the current RA group of L_{TR}, the overlap could occur. This is because an object in the current RA group (k_x) for a keyword k_x may have a low $f_{TR}(k_y, o.doc)$ for another keyword k_y, and consequently, the keyword component score of the object can be lower than those of the objects that have not yet been processed. For example, in Figure 7, even though o_5 is included in the current RA group (‘food’) and o_3 in the next RA group (‘food’), $f_{TR}(q.keywords, o_3)$ is higher than $f_{TR}(q.keywords, o_5)$ (see Figure 10).

By using this generalization, we propose a *deferred random access technique* that reduces random accesses required by External TA. Due to the overlap, the spatial (keyword) component scores of the objects in the current RA group may be lower than $\overline{S_{SP}}$ ($\overline{S_{TR}}$), and the combined scores of these objects are lower than the threshold value. From this property, we can reduce the number of random accesses by deferring random accesses until they are necessary. When the object o is included only in the current RA group of L_i ($i =$ SP or TR), we first simply compute the highest possible combined score of o, which we denote by $\overline{f}(q, o)$, instead of randomly accessing o. $\overline{f}(q, o)$ is computed from (1) by taking $\overline{S_j}$ of the other list L_j ($i \neq j$) as the component score. If $\overline{f}(q, o)$ is higher than the threshold value, o may belong to the top-k result at this iteration, and we randomly access o from L_j ($i \neq j$) to obtain the precise component score of o. If not, o cannot belong to the top-k result at this iteration, and we do not need to access the objects; thus, we defer random access to these objects—adding them to the next RA group for the next iteration.

Figure 13 shows Generalized External TA. The inputs are the lists L_{SP}, L_{TR}, and a top-k spatial keyword query q. The output is the list of top-k objects with the highest combined scores in (1). The algorithm initializes *CSLowerThanThreshold* and *CSHigherThanThreshold*. It also initializes *HPCSLowerThanThreshold*, the objects whose highest possible combined scores (HPCS) are lower than or equal to the threshold value. We only explain Steps 1, 3, and 4-1 because Generalized External TA is different from External TA only in these steps. In Step 1, the algorithm retrieves the current RA groups CG_{SP} and CG_{TR} from L_{SP} and L_{TR} in proportion to the user preference. In Step 3, the algorithm merges the objects in *HPCSLowerThanThreshold* into CG_{SP} and CG_{TR}. Here, if each object in *HPCSLowerThanThreshold* was retrieved from L_{SP}, it is merged into CG_{SP}; if each object was retrieved from L_{TR}, it is merged into CG_{TR}. In Step 4-1, if o is retrieved from both lists L_{SP} and L_{TR}, the algorithm directly computes the combined score; if o is retrieved from only one list L_i, the algorithm does the following: If $\overline{f}(q, o)$ is higher than the threshold value, the algorithm randomly accesses the other list L_j ($i \neq j$) to obtain the component score of o; if not, the algorithm stores o in *HPCSLowerThanThreshold* for the next iteration.

Theorem 2 *Generalized External TA correctly finds the top-k results.*

Proof Unlike External TA, Generalized External TA excludes the objects whose $\overline{f}(q, o)$ are lower than or equal to the threshold value from *CSHigherThanThreshold* at this iteration. Thus, we only need to prove that such objects cannot belong to *CSHigherThanThreshold*. If $\overline{f}(q, o)$ is lower than or equal to the threshold value, so is the combined score of o since $\overline{f}(q, o)$ is the highest possible combined score of o. Since the combined scores of the objects in *CSHigherThanThreshold* must be higher than the threshold value, o cannot belong to *CSHigherThanThreshold*. □

Example 7 In Example 5, the step for calculating the combined scores of the objects in the current RA group is modified in Generalized External TA as follows:

– $f(q, o_1) = 0.643$ (from (1) with $f_{SP}(q.loc, o_1) = 0.6$ and $f_{TR}(q.loc, o_1) = 0.686$)

```
Algorithm Generalized External TA:
1:  Input: (1) $L_{SP}$, (2) $L_{TR}$, (3) $q$: a top-$k$ spatial keyword query
2:  Output: Result: the list of top-$k$ objects with the highest combined scores $f()$
3:  CSLowerThanThreshold := { };          /* initialize CSLowerThanThreshold */
4:  CSHigherThanThreshold:= { };          /* initialize CSHigherThanThreshold */
5:  HPCSLowerThanThreshold := { };        /* initialize HPCSLowerThanThreshold */
6:  REPEAT
      /* Step 1: Retrieve the current RA groups and compute the threshold value Threshold */
7:    Retrieve the current RA groups $CG_{SP}$ and $CG_{TR}$ from $L_{SP}$ and $L_{TR}$ in proportional to
      the user preference;
8:    Compute Threshold by plugging $\overline{S_{SP}}$ and $\overline{S_{TR}}$ into Eq. (1) as the component scores;
      /* Step 2: Examine the objects in CSLowerThanThreshold */
9:    … // These lines are omitted because they are the same as line 8~13 of External TA in Fig. 11
      /* Step 3: Merge the objects in HPCSLowerThanThreshold into the current RA groups */
10:   FOR each $o \in$ HPCSLowerThanThreshold DO BEGIN
11:       IF $o$ was retrieved from $L_{SP}$ THEN
12:           $CG_{SP} := CG_{SP} \cup \{o\}$;
13:       ELSE /* $o$ was retrieved from $L_{TR}$ */
14:           $CG_{TR} := CG_{TR} \cup \{o\}$;
15:       HPCSLowerThanThreshold := HPCSLowerThanThreshold – $\{o\}$;
16:       END
      /* Step 4: Examine the objects in the current groups */
      /* $CG_{SP}$ and $CG_{TR}$ are in the order of the object ID */
17:   $i$ := the first element in $CG_{SP}$; $j$ := the first element in $CG_{TR}$;
18:   WHILE objects are remaining in either $CG_{SP}$ or $CG_{TR}$ DO BEGIN
          /* Step 4-1: Compute the combined score of each object $o$ */
19:       IF object ID($i$) = object ID($j$) THEN BEGIN
20:       … // These lines are omitted because they are the same as line 8~13 of External TA in Fig. 11
21:       ELSE IF object ID($i$) > object ID($j$) THEN BEGIN
22:           $o := j$;                   /* $o$ is retrieved only from $L_{TR}$ */
23:           Compute $\overline{f}(q, o)$;
24:           IF $\overline{f}(q, o) >$ Threshold THEN BEGIN
25:           … // These lines are omitted because they are the same as line 8~13 of External TA in Fig. 11
26:           ELSE /* $\overline{f}(q, o) \leq$ Threshold */
27:               HPCSLowerThanThreshold := HPCSLowerThanThreshold $\cup \{o\}$;
28:           Advance $j$ to the next object in $CG_{TR}$;
29:           END
30:       ELSE BEGIN /* object ID($i$) < object ID($j$) */
31:           $o := i$;                   /* $o$ is retrieved only from $L_{SP}$ */
32:           Compute $\overline{f}(q, o)$;
33:           IF $\overline{f}(q, o) >$ Threshold THEN BEGIN
34            … // These lines are omitted because they are the same as line 8~13 of External TA in Fig. 11
35:           ELSE /* $\overline{f}(q, o) \leq$ Threshold */
36:               HPCSLowerThanThreshold := HPCSLowerThanThreshold $\cup \{o\}$;
37:           Advance $i$ to the next object in $CG_{SP}$;
38:           END
          /* Step 4-2: Store the object according to its combined score */
39:       … // These lines are omitted because they are the same as line 8~13 of External TA in Fig. 11
40: UNTIL |CSHigherThanThreshold| $\geq k$
    /* Step 5: Retrieve the top-$k$ results */
41: RETURN top-$k$ objects in CSHigherThanThreshold;
```

Figure 13 Generalized External TA.

- $\overline{f}(q, o_2) = 0.605$ (from (1) with $f_{SP}(q.loc, o_2) = 0.7$ and $\overline{S_{TR}} = 0.510$)
- $\overline{f}(q, o_4) = 0.3685$ (from (1) with $\overline{S_{SP}} = 0.5$ and $f_{TR}(q.loc, o_4) = 0.237$)

We randomly access o_2 to obtain the component score since $\overline{f}(q, o_2) >$ threshold value, but do not access o_4.

5 Performance evaluation

5.1 Experimental data and environment

In this section, we compare the index size and query performance of RASIM with those of the IR-tree method [10]. The index size of RASIM is the sum of the size of the RA spatial index and that of the RA inverted index while the index size of the IR-tree method is that of the IR-tree itself. For measuring the query performance, we use the wall clock time and the number of page accesses. We use three data sets: *DataSet1*, *DataSet2*, and *DataSet3*. In *DataSet1*, the spatial data contains two-dimensional real spatial objects for buildings located in Seoul, and the text data is from Web pages collected through Web crawling. In *DataSet2*, the spatial data are generated randomly, and the text data are from Web pages collected through Web crawling. We generate five sets with different sizes for *DataSet2*: 20K, 40K, 60K, 80K, and 100K. The data sets were generated by randomly selecting a Web page for each spatial object. *DataSet3* is created combining texts from 20 Newsgroups dataset[4] and locations from LA streets.[5] This is the same dataset used by Cong et al. [10]. We use *DataSet2* in the experiment to measure the performance as the data size is varied and *DataSet1* and *DataSet3* in the other experiments. Table 3 shows the characteristics of the data sets used in the experiment.

We generate four query sets, in which the number of keywords is 1, 2, 3, and 4, respectively. Each query set is composed of 100 queries, and each keyword in a query is randomly selected. We present the average wall clock time and the numbers of object and page accesses of the queries.

We conducted all the experiments on a Pentium4 3.6GHz Linux PC with 1.5GB of main memory. To avoid the buffering effect of the LINUX system and to guarantee actual disk I/O's, we used raw disks for storing data and indexes. We implemented both RASIM and the IR-tree method [10] using the MBR-MLGF [26] and the inverted index implemented in the Odysseus DBMS [32, 33]. The page size for data and indexes is set to 4,096 bytes.

The MBR-MLGF is an RA spatial index based on the MLGF [30]. An MLGF is a balanced tree and consists of a multilevel directory and leaf nodes. An entry in a leaf-level node contains a region vector and a pointer to an object; an entry in a non-leaf-level node consists of the region vector and the pointer to a child node. In the *n*-dimensional case, a region vector consists of *n* hash values that uniquely identify the region. The MBR-MLGF is a variation of the MLGF that represents each region vector as an MBR containing the objects in the region.

[4] http://people.csail.mit.edu/jrennie/20Newsgroups

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

Table 3 Characteristics of the data sets.

Data sets	*DataSet1*	*DataSet2*	*DataSet3*
Total number of objects	78,260	100,000	131,461
Maximum number of postings per keyword	55,949	78,463	131,461
Average number of postings per keyword	89	82	168
Total number of unique words in the data set	206,636	318,916	114,831
Total number of words in the data set	18,397,075	26,161,111	19,278,878

The MBR-MLGF and R-tree family [4, 14, 25] differ in the way they cluster the objects. While the MBR-MLGF clusters the objects in the transformed space, the R-tree family clusters the objects in the original space. It has been shown through experiments [27] that clustering effects of the objects in the MBR-MLGF are comparable to or better than those in the R*-tree. The IR-tree method is based on the R-tree, but can also be applied to the MBR-MLGF in a similar way. Thus, for the sake of fairness, we compare the performance of RASIM with that of the IR-tree method that uses the MBR-MLGF.

5.2 Results of the Experiments

5.2.1 Index size

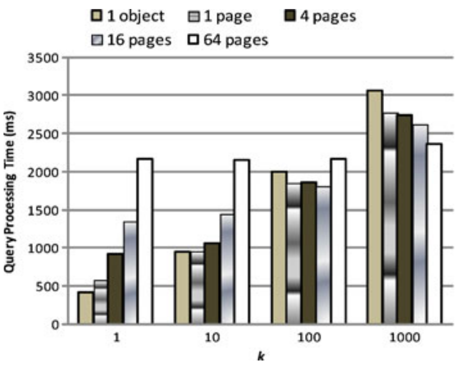
Table 4 shows the sizes of the indexes. The index size of RASIM is reduced by 1.43$\sim$1.85 times compared with that of the IR-tree method. This result supports our discussion in Section 2.

5.2.2 Query processing performance

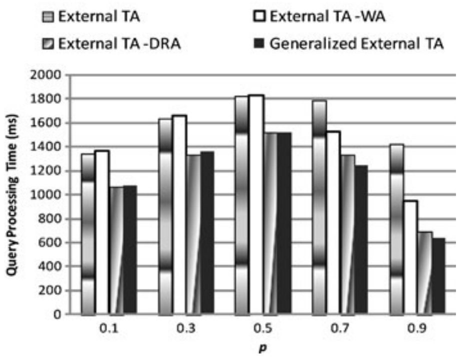
Performance analysis of RASIM Figure 14a shows the performance of RASIM with different group sizes as k is varied. Here, the user preference $p = 0.5$ and the number of keywords in the query $nKeywords = 1$. We observe that a small (large) group size is more efficient for a small (large) k. Specifically, for the case of $k = 1$, query performance for the group size of one object is the most efficient; for the case of $k = 1000$, that for the group size of 64 pages is the most efficient. Figure 14b shows overall performance enhancement of Generalize External TA as p is varied. Here, $k = 100$ and $nKeywords = 1$. In Figure 14, *External TA-WA (External TA-DRA)* is the method that applies weighted access (deferred random access) to External TA; Generalized External TA is the method that applies both of them to External TA. We observe that weighted access enhances the query performance by 0.98$\sim$1.51, and deferred random access by 1.20$\sim$2.06. Overall, Generalized External TA enhances the performance of External TA by 1.20$\sim$2.22. Henceforth, we use Generalized External TA as the query processing algorithm of RASIM for the comparison with the IR-tree method.

Table 4 Sizes of indexes (KB).

Method	*DataSet1*	*DataSet2*	*DataSet3*
RASIM	654,908	1,170,852	870,668
IR-tree method	981,420	1,677,320	1,611,156



(a) query performance of RASIM with different group size

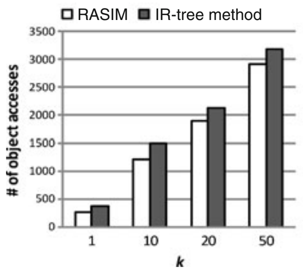


(b) overall performance enhancement of Generalized External TA

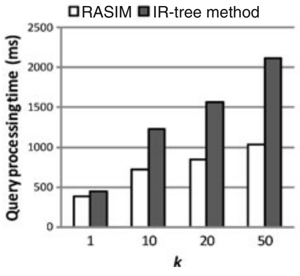
Figure 14 The query performance analysis of RASIM.

Comparison with the IR-tree method Figure 15 shows the performance as k is varied. Here, $p = 0.5$ and $nKeywords = 1$. We observe that the number of object accesses of RASIM is reduced by 1.09∼1.42 times, the query processing time by 1.15∼2.03 times, and the number of page accesses by 1.28∼2.17 times as compared to those of the IR-tree method. There are three reasons for this performance advantage. (i) The pruning power of RASIM, which inherits from TA [11], is comparable to or better than that of the IR-tree method, which inherits from Incremental NN [17]. This result is evidenced by the number of object accesses measured in Figure 15a. (ii) RASIM obviates extra accesses to redundantly stored part of the inverted index (i.e., access to the inverted indexes on internal nodes) in the IR-tree method. (iii) RASIM clusters the text index in one file while the IR-tree method stores it in multiple files over different nodes resulting in multiple fragmented text indexes. (ii) and (iii) are supported by the fact that while the number of object accesses in RASIM is slightly less than that in the IR-tree method as in Figure 15a, the number of page accesses in RASIM is much less than that in the IR-tree method as in Figure 15c.

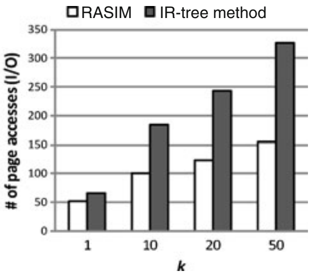
Figure 16 shows the performance as p is varied. Here, $k = 10$ and $nKeywords = 1$. We observe that the query processing time of RASIM is reduced by 1.08∼3.22 times and the number of page accesses by 1.31∼3.65. We note that the gap of



(a) the number of object accesses.

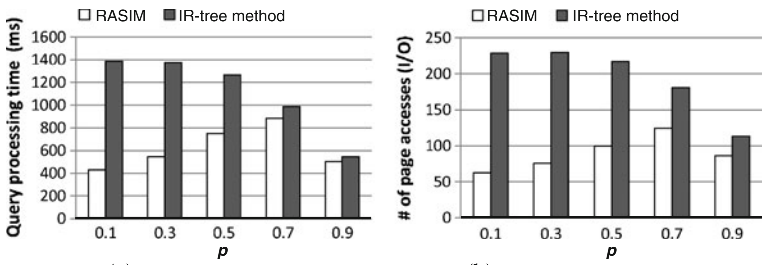


(b) query processing time.



(c) the number of page accesses.

Figure 15 The query performance as k is varied.



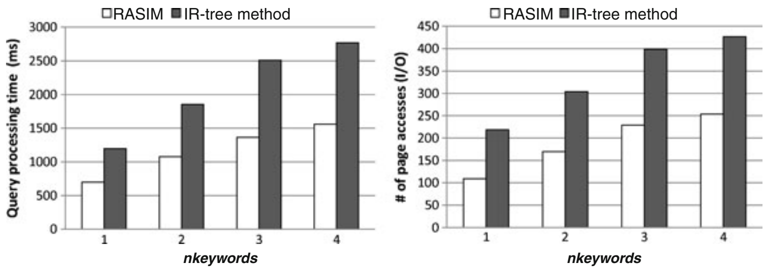
(a) query processing time. (b) the number of page accesses

Figure 16 The query performance as p is varied.

query performance between RASIM and the IR-tree method becomes larger as p is decreased (i.e., less weight on the spatial component score than on the keyword component score). The query performance of the IR-tree method is degraded as p is decreased because each leaf node of the IR-tree contains objects clustered according to spatial proximity, but the IR-tree method cannot take advantage of the effects of the clustering when a low weight is assigned to spatial proximity. In contrast, the performance of RASIM is relatively less affected by p since RASIM can control the group size of each list according to p.

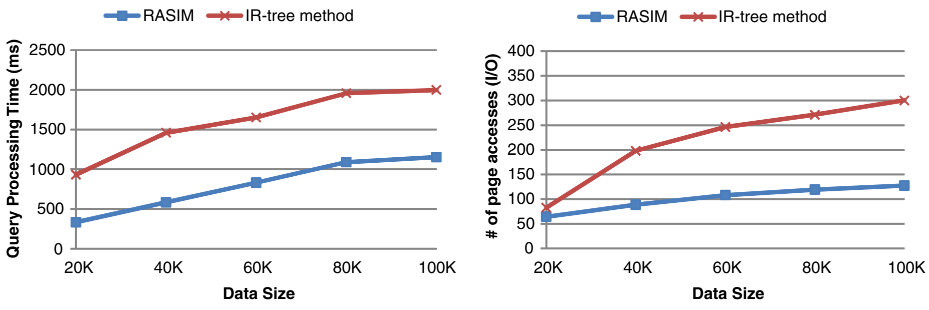
Figure 17 shows the performance as $nKeywords$ is varied. Here, $k = 10$ and $p = 0.5$. We observe that the query processing time of RASIM is reduced by 1.72∼1.84 times and the number of page accesses by 1.69∼2.00. Figure 18 shows the performance as the data set size increases. Here, $k = 10$, $p = 0.5$, and $nKeywords = 1$. We observe that the performance advantage of RASIM over the IR-tree method stays relatively constant regardless of the data set size.

Experiments on DataSet3 We have performed the same extensive experiments on *DataSet3* as on *DataSet1*. Figure 19 shows the performance as k is varied; Figure 20 shows the performance as p is varied; Figure 21 shows the performance as $nKeywords$ is varied. For each experiment, the default parameters are as follows: $k = 10$, $p = 0.5$, and $nKeywords = 1$. The results are consistent with those for *DataSet1*.



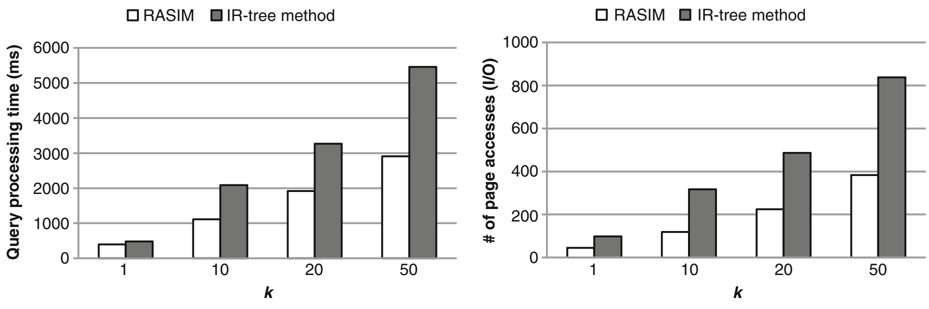
(a) query processing time. (b) the number of page accesses

Figure 17 The query performance as $nKeywords$ is varied.



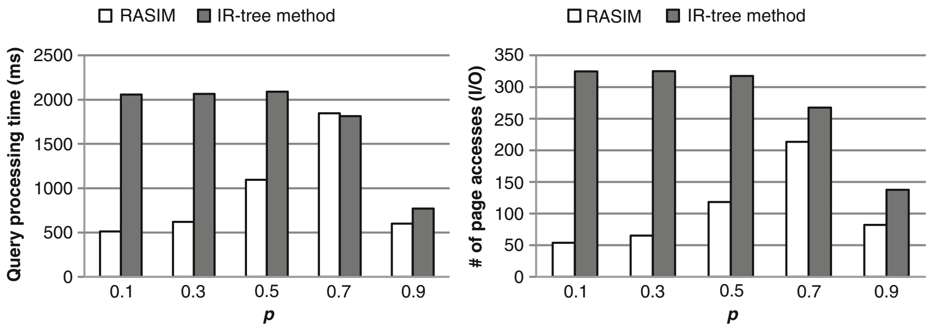
(a) query processing time. (b) the number of page accesses

Figure 18 The query performance as the data set is varied.



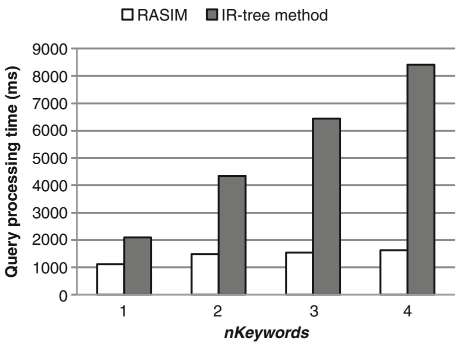
(a) query processing time. (b) the number of page accesses

Figure 19 The query performance as *k* is varied on *DataSet3*.

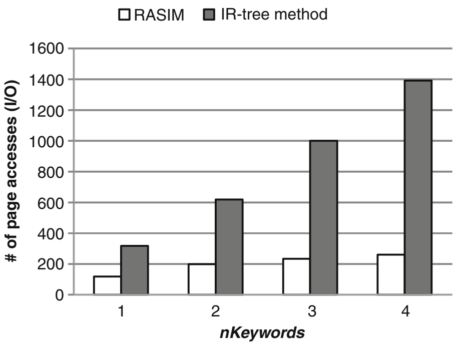


(a) query processing time. (b) the number of page accesses

Figure 20 The query performance as *p* is varied on *DataSet3*.



(a) query processing time.



(b) the number of page accesses

Figure 21 The query performance as *nKeywords* is varied on *DataSet3*.

6 Conclusions

We have proposed a new separate index method, called Rank-Aware Separate Index Method (RASIM). RASIM is the first research work that supports top-*k* pruning while using the separate index method. The key property of RASIM is that each index is clustered by two different orders—one for top-*k* pruning and the other for efficient merging. We have shown that this property can be obtained through partitioning. RASIM partitions the set of objects in each index to a set of RA groups that contain objects with similar scores. Based on the notion of the RA group, we have proposed two query processing algorithms: (i) External TA, which supports top-*k* pruning in the unit of the RA group mapped to a disk page and (ii) Generalized External TA, which enhances the performance of External TA by exploiting the special properties of the RA groups. We have proved the correctness of the proposed algorithms in Theorems 1 and 2.

We have shown through experiments that RASIM is more efficient than the IR-tree method in terms of both storage and query processing time. The IR-tree method is the prevailing method to date that supports top-*k* pruning for top-*k* spatial keyword queries. RASIM improves storage efficiency since the IR-tree method stores part of the inverted index redundantly for top-*k* pruning, but RASIM obviates this redundant data structure. Experimental results show that the size of the index in RASIM is by up to 1.85 times smaller than that of the IR-tree method. RASIM improves the query performance since (i) it maintains pruning power comparable to or better than the IR-tree method, (ii) obviates extra accesses to redundantly stored part of the index, and (iii) clusters the text index in one file while the IR-tree method stores it in multiple files over different nodes resulting in multiple fragmented text indexes. Experimental results show that the query processing time of RASIM is improved by 1.08~3.22 times compared with that of the IR-tree method as the parameters *k*, user preference, the number of keywords, and data set size are varied.

A very important observation on RASIM is that, since it is based on the separate index approach, it inherits the advantages of scalability for accommodating new data types and flexibility of maintaining each index independently. Thus, we can utilize the RASIM approach for integrating other (possibly new) data types besides spatial and text types. In this regard, we expect it to trigger many interesting research investigations in processing *top-k multiple-type integrated queries*. We leave this as a further study.

Acknowledgements This work was partially supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2012R1A2A2A01012605) and was also partially supported by the MKE(The Ministry of Knowledge Economy), Korea and Microsoft Research, under IT/SW Creative research program supervised by the NIPA(National IT Industry Promotion Agency) (NIPA-2010-C1810-1002-0010) and partially supported by Microsoft Research Asia through KAIST-Microsoft Research Collaboration Center (KMCC).

References

1. Anh, V., Moffat, A.: Impact transformation: effective and efficient Web retrieval. In: Proc. ACM SIGIR Int'l Conf. on Research and Development in Information Retrieval, pp. 3–10 (2002)
2. Asadi, S., Zhou, X., and Yang, G.: Using local popularity of Web resources for geo-ranking of search engine results. World Wide Web J. **12**(2), 149–170 (2009)
3. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. ACM Press, Addison-Wesley (1999)
4. Beckmann, N., Kriegel, H., Schneider, R., Seeger, B.: The R*-Tree: an efficient and robust access method for points and rectangles. In: Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp. 322–331 (1990)
5. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Comput. Netw. ISDN Syst. **30**(1–7), 107–117 (1998)
6. Brown, E.W.: Fast evaluation of structured queries for information retrieval. In: Proc. ACM Int'l SIGIR Conf. on Research and Development in Information Retrieval, pp. 30–38 (1995)
7. Chang, Y., Bergman, L., Castelli, V., Li, C., Lo, M., Smith, J.: The ONION technique: indexing for linear optimization queries. In: Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp. 391–402 (2000)
8. Chaudhuri, S., Ramakrishnan, R., Weikum, G.: Integrating DB and IR technologies: what is the sound of one hand clapping? In: Proc. Conf. on Innovative Data Systems Research (CIDR), pp. 1–12 (2005)
9. Chen, Y., Suel, T., Markowetz, A.: Efficient query processing in geographic Web search engines. In: Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp. 277–288 (2006)
10. Cong, G., Jensen, C., Wu, D.: Efficient retrieval of the Top-k most relevant spatial web objects. In: Proc. 35th Int'l Conf. on Very Large Data Bases (VLDB), pp. 754–765 (2009)
11. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proc. 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), pp. 102–113 (2001)
12. Felipe, I., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: Proc. 24th Int'l Conf. on Data Engineering (ICDE), IEEE, pp. 656–665 (2008)
13. Guo, L., Shanmugasundaram, J., Beyer, K., Shekita, E.: Efficient inverted lists and query algorithms for structured value ranking in update-intensive relational databases. In: Proc. 21st Int'l Conf. on Data Engineering (ICDE), IEEE, pp. 298–309 (2005)
14. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp. 47–57 (1984)
15. Hariharan, R., Hore, B., Li, C., Mehrotra, S.: Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) systems. In: Proc. 19th Int'l Conf. on Scientific and Statistical Database Management (SSDBM), p. 16 (2007)
16. Harper, S., Chen, A.: Web accessibility guidelines: a lesson from the evolving Web. World Wide Web J. **15**(1), 61–88 (2012)
17. Hjaltason, G., Samet, H.: Distance browsing in spatial databases. ACM Trans. Database Syst. **24**(2), 265–318 (1999)
18. Ilyas, I., Beskales, G., Soliman, M.: A survey of Top-*K* query processing techniques in relational database systems. ACM Comput. Surv. **40**(4), 11 (2008)
19. Li, Z., Lee, K., Zheng, B., Lee, W., Lee, D., Wang, X.: IR-Tree: an efficient index for geographic document search. IEEE Trans. Knowl. Data Eng. **23**(4), 585–599 (2011)

20. Long, X., Suel, T.: Optimized query execution in large search engines with global page ordering. In: Proc. 29th Int'l Conf. on Very Large Data Bases (VLDB), pp. 129–140 (2003)
21. Martins, B., Silva, M., Adnrade, L.: Indexing and ranking in Geo-IR systems. In: Proc. 2nd Int'l Workshop on Geo-IR(GIR), ACM SIGIR, pp. 31–34 (2005)
22. Masutani, O., Iwasaki, H.: BEIRA: an area-based user interface for map services. World Wide Web J. **12**(1), 51–68 (2009)
23. Park, D., Kim, H.: An enhanced technique for k-Nearest neighbor queries with non-spatial selection predicates. Multimedia Tools and Application Archive **19**(1), 79–103 (2003)
24. Rocha-Junior, J., Gkorgkas, O., Jonassen, S., Norvag, K.: Efficient processing of Top-k spatial keyword queries. In: Proc. 12th Intl Symposium on Spatial and Temporal Databases (SSTD), pp. 205–222 (2011)
25. Sellis, T., Roussopoulos, N., Faloutsos, C.: The R+-Tree: a dynamic index for multi-dimensional objects. In: Proc. 13th Int'l Conf. on Very Large Data Bases (VLDB), pp. 507–518 (1987)
26. Song, J., Whang, K., Lee, Y., Kim, S.: Spatial join processing using corner transformation. IEEE Trans. Knowl. Data Eng. **11**(4), 688–698 (1999)
27. Song, J., Whang, K., Lee, Y., Lee, M., Han, W., Park, B.: The clustering property of corner transformation for spatial database applications. Inf. Softw. Technol. **44**(7), 419–429 (2002)
28. Vaid, S., Jones, C., Joho, H., Sanderson, M.: Spatio-textual indexing for geographical search on the Web. In: Proc. 9th International Symposium on Spatial and Temporal Databases (SSTD), pp. 218–235 (2005)
29. Weikum, G.: DB&IR: both sides now. In: Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp. 25–30 (2007)
30. Whang, K., Krishnamurthy, R.: The multilevel grid file: a dynamic hierarchical multidimensional file structure. In: Proc. Int'l Symposium on Database Systems for Advanced Applications (DAS-FAA), pp. 449–459 (1991)
31. Whang, K., Park, B., Han, W., Lee, Y.: Inverted index storage structure using subindexes and large objects for tight coupling of information retrieval with database management systems. United States Patent 6349308. Appl. No. 09/250,487, 15 Feb. 1999 (2002)
32. Whang, K., Lee, M., Lee, J., Kim,M., Han,W.: Odysseus: a high-performance ORDBMS tightly-coupled with IR features. In: Proc. 21st Int'l Conf. on Data Engineering (ICDE), IEEE, pp. 1104–1105 (2005). This paper received the Best Demonstration Award
33. Whang, K., Lee, J., Kim, M., Lee, M., Lee, K., Han, W., Kim, J.: Tightly-coupled spatial database features in the Odysseus/OpenGIS DBMS for high-performance. GeoInformatica **14**(4), 425–446 (2010)
34. Zhou, Y., Xie, X., Wang, C., Gong, Y., Ma, W.: Hybrid index structures for location-based Web search. In: Proc. 14th ACM Conf. on Information and Knowledge Management (CIKM), pp. 155–162 (2005)
35. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Comput. Surv. **38**(2), 1–56 (2006)