World Wide Web (2016) 19:1051–1075
DOI 10.1007/s11280-015-0373-x

Scalable and efficient processing of top-*k* multiple-type integrated queries

Hyuk-Yoon Kwon[1] · **Kyu-Young Whang**[1]

Received: 23 November 2014 / Revised: 14 August 2015 /
Accepted: 1 November 2015 / Published online: 19 November 2015

Abstract In this paper, we define a new class of queries, the *top-k multiple-type integrated query* (simply, *top-k MULTI query*). It deals with multiple data types and finds the information in the order of relevance between the query and the object. Various data types such as spatial, textual, and relational data types can be used for the top-k MULTI query. The top-k MULTI query distinguishes itself from the traditional top-k query in that the component scores to calculate final scores are determined dependent of the query. Hence, each component score is calculated only when the query is given for each data type rather than being calculated apriori as in the top-k query. As a representative instance, the traditional top-k spatial keyword query is an instance of the top-k MULTI query. It deals with the spatial data type and text data type and finds the information based on spatial proximity and textual relevance between the query and the object, which is determined only when the query is given. In this paper, we first define the top-k MULTI query formally and define a new specific instance for the top-k MULTI query, the *top-k spatial-keyword-relational(SKR) query*, by integrating the relational data type into the traditional top-k spatial keyword query. Then, we investigate the processing approaches for the top-k MULTI query. We discuss the scalability of those approaches as new data types are integrated. We also devise the processing methods for the top-k SKR query. Finally, through extensive experiments on the top-k SKR query using real and synthetic data sets, we compare efficiency of the methods in terms of the query performance and storage.

Keywords Top-k multiple-type integrated query · Top-k spatial-keyword-relational query · Scalability · Efficiency

✉ Kyu-Young Whang
kywhang@cs.kaist.ac.kr

Hyuk-Yoon Kwon
hykwon@mozart.kaist.ac.kr

[1] Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea

1 Introduction

1.1 Background

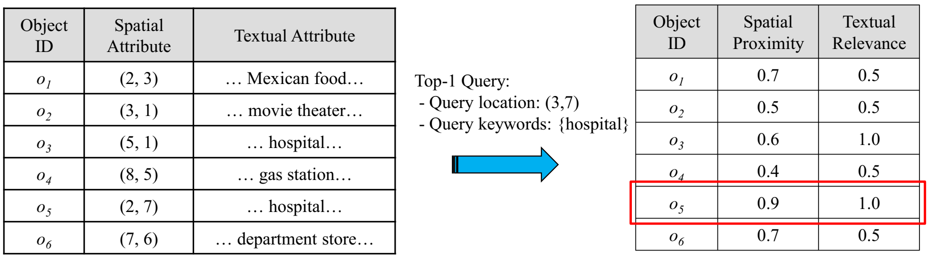
In the era of big data, new data types are frequently generated, and the data integrating multiple types are commonly encountered. Thus, multiple data types such as locations, texts, or structured data are frequently involved in queries searching for the information from data. For example, we search for the information that is close to the user using the location. We also often search for the information that the user wants using the keywords. We may also search for the information using a structured type such as the name or phone number. For convenience, we integrate these queries on different data types in one query form. We define such queries as the *top-k MULtiple-Type Integrated query* (simply, *top-k MULTI query*). The top-k MULTI query finds the information in the order of relevance between the query and the object.

A representative instance of the top-k MULTI query is the top-k spatial keyword query [3, 8, 12]. The top-k spatial keyword query searches for the information based on spatial proximity and textual relevance between the query and the object. Figure 1 shows an example top-k spatial keyword query. The object consists of a spatial attribute and a textual attribute. The query is "Find the top-1 object that is proximate to the location (3, 7) and that is relevant to the keyword 'hospital'." The query finds o_5 as the result based on spatial proximity and textual relevance.

There have been many research efforts on top-k spatial keyword query processing [3, 8–10, 12, 20, 21]. The efforts can be classified into two categories according to the index structures used: 1) the hybrid index approach [3, 10, 12, 20, 21] and 2) the separate index approach [8]. The former builds the spatial index and textual index in a combined index. It maintains both the spatial attribute and textual attribute for each object together; thus, it does not need to merge the objects retrieved from the individual indexes. The latter maintains the spatial index and the textual index separately and merges the objects that are retrieved from each of them. The index structure of the separate index approach is simpler than that of the hybrid index approach; thus, it is easier to maintain.

1.2 Our contributions

In this paper, we make the following four contributions. First, we define the top-k MULTI query formally. It deals with multiple data types and finds the information in the order of relevance between the query and the object. The top-k MULTI query is practically important since we can find relevant results even if we do not know the exact values to find. We clarify



Object ID	Spatial Attribute	Textual Attribute
o_1	(2, 3)	… Mexican food…
o_2	(3, 1)	… movie theater…
o_3	(5, 1)	… hospital…
o_4	(8, 5)	… gas station…
o_5	(2, 7)	… hospital…
o_6	(7, 6)	… department store…

Top-1 Query:
- Query location: (3,7)
- Query keywords: {hospital}

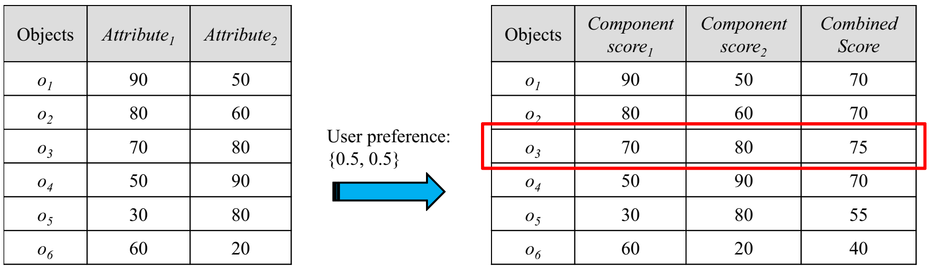
Object ID	Spatial Proximity	Textual Relevance
o_1	0.7	0.5
o_2	0.5	0.5
o_3	0.6	1.0
o_4	0.4	0.5
o_5	0.9	1.0
o_6	0.7	0.5

Figure 1 An example top-k spatial keyword query

the difference between the top-k MULTI query and the traditional top-k query [7]. The component scores for the former are determined dependent of the query while those for the latter are determined independent of the query. That is, the former calculates the component score from the query and the object when the query is given while the latter uses the attribute value of the object as the component score itself regardless of the query. Let us consider two relational attributes as an example. One attribute represents the score for ‘Math’ and the other one for ‘English’. The top-k MULTI query finds the top-k students whose scores for ‘Math’ and ‘English’ are *the closest to the specific given scores*; the top-k query finds the top-k students whose combined scores obtained by summing up the attribute values for ‘Math’ and ‘English’ are *the highest*. Thus, query processing for these two classes of queries is quite different.

Example 1 Figure 2a shows that, in a top-k query, the component scores of the objects are the attribute values themselves. As the top-1 result, the query returns o_3. Figure 2b shows that, in a top-k MULTI query, the component scores of the objects are derived from the given query values and the attribute values. As the top-1 result, the query returns o_2.

Second, to show the importance of the top-k MULTI query, we define a new specific instance of the top-k MULTI query, i.e., the *top-k spatial-keyword-relational(SKR) query*, by integrating the relational data type to the top-k spatial keyword query. *Relational closeness* is a new ranking measure for the relational attribute in the top-k SKR query and represents relevance between the relational value of the object and the relational value of the query. This feature distinguishes the top-k SKR query from the top-k query. In the top-k query, the relational attribute value itself is used for the component score as explained in Example 1.

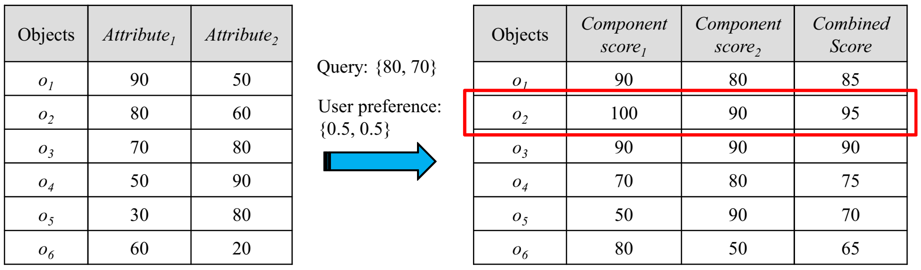


Objects	*Attribute$_1$*	*Attribute$_2$*
o_1	90	50
o_2	80	60
o_3	70	80
o_4	50	90
o_5	30	80
o_6	60	20

User preference:
{0.5, 0.5}

Objects	*Component score$_1$*	*Component score$_2$*	*Combined Score*
o_1	90	50	70
o_2	80	60	70
o_3	70	80	75
o_4	50	90	70
o_5	30	80	55
o_6	60	20	40

(a) Score calculation in top-k queries.



Objects	*Attribute$_1$*	*Attribute$_2$*
o_1	90	50
o_2	80	60
o_3	70	80
o_4	50	90
o_5	30	80
o_6	60	20

Query: {80, 70}
User preference:
{0.5, 0.5}

Objects	*Component score$_1$*	*Component score$_2$*	*Combined Score*
o_1	90	80	85
o_2	100	90	95
o_3	90	90	90
o_4	70	80	75
o_5	50	90	70
o_6	80	50	65

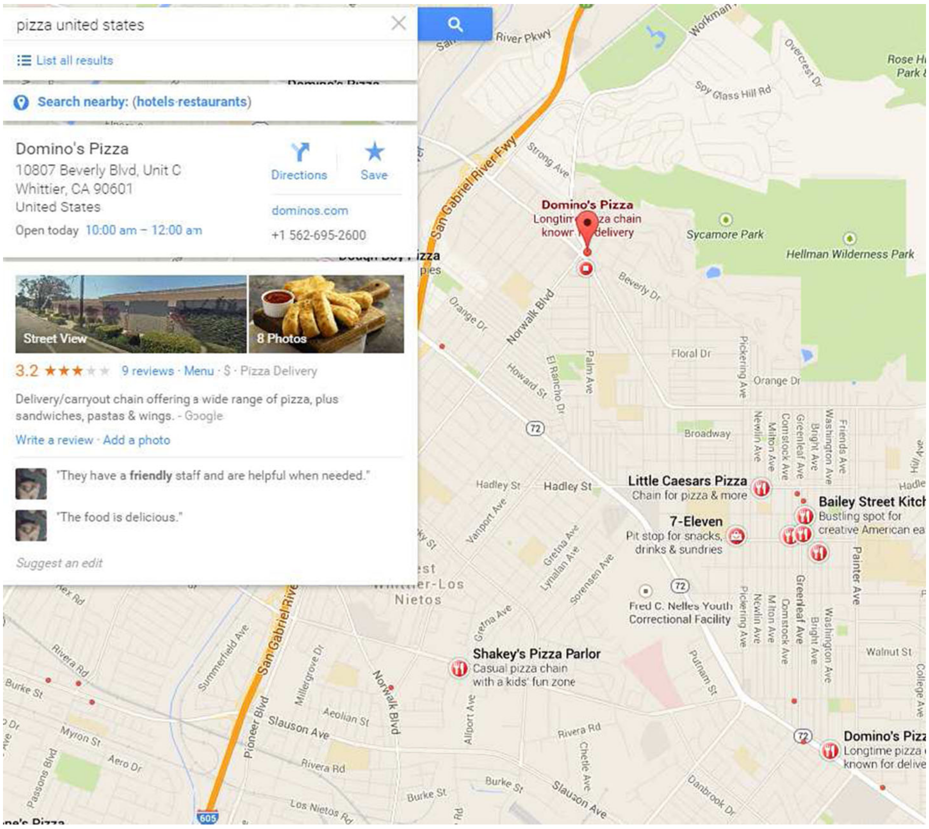
(b) Score calculation in top-k MULTI queries.

Figure 2 The comparison between the top-k MULTI query and the top-k query

Relational closeness can be useful when finding the objects that have relational values similar to the given relational query value. Especially, even if the user does not know the exact values of objects to find, we can allow finding the most relevant objects to the query value based on relational closeness. To show the usefulness of the top-k SKR query, we introduce two real-life applications.

Application 1 Figure 3 *shows Google Maps. Let us consider a query "Find a restaurant that is close to the current location, sells 'pizza', and has the number of reviews near to 10." In the query, there are three measures to find the result relevant to the query: 1) spatial proximity, 2) textual relevance, and 3) relational closeness. That is, spatial proximity is calculated by the distance between the location of the restaurant and the location of the user; textual relevance is calculated by the relevance between the textual description of the restaurant and a query keyword 'pizza'; relational closeness is calculated by the difference between the number of reviews of the restaurant and 10.*

Application 2 Figure 4 *shows data for patients with infectious diseases. Each record consists of the location of a patient, disease name, and infection year. Let us consider a query*


pizza united states

Domino's Pizza
10807 Beverly Blvd, Unit C
Whittier, CA 90601
United States
Open today 10:00 am – 12:00 am

dominos.com
+1 562-695-2600

3.2 ★★★ 9 reviews · Menu · $ · Pizza Delivery

Delivery/carryout chain offering a wide range of pizza, plus sandwiches, pastas & wings. - Google

"They have a **friendly** staff and are helpful when needed."

"The food is delicious."

Figure 3 Google Maps

Patient ID	Location	Disease	Year
A	(20,20)	Diphtheria	2011
B	(30,20)	Diphtheria	2005
C	(70,50)	SARS	2002
D	(15,60)	Typhoid	2007
E	(25,15)	SARS	2003
F	(10,10)	Tuberculosis	2009

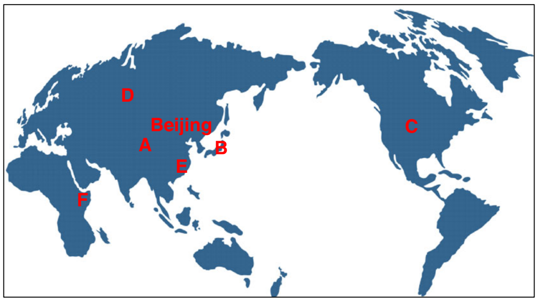
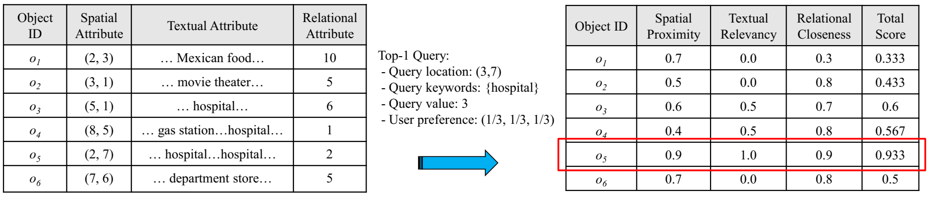


Figure 4 Data for patients with infectious diseases

"Find a patient whose residence is close to 'Beijing' (25, 25), disease name is 'SARS', and infection date is around 2002." Here, the difference between the infection year and 2003 is used as the relational closeness. In this query, E would be the top-1 result.

Example 2 Consider a top-*k* SKR query to find the top-1 object that is proximate to the query location (3, 7), relevant to the query keyword {hospital}, and close to the relational query value 3. Here, user preference is given as follows: 1/3 for spatial proximity, 1/3 for textual relevance, and 1/3 for relational closeness. Figure 5 shows an example top-*k* SKR query. For the given query, we first calculate spatial proximity, textual relevance, and relational closeness of each object. Then, we calculate their total scores considering user preference. As the result, we return o_5.

Third, we investigate processing approaches for the top-*k* MULTI query: the hybrid index and separate index approaches. We discuss the scalability of those approaches as the new data types are integrated. Scalability of the approach is the key issue for top-*k* MULTI query processing since many data types can be integrated for the top-*k* MULTI query. The hybrid index approach builds all the indexes used for the top-*k* MULTI query in an integrated form; i.e., it builds them in multi-level indexes. As a result, the indexes built in low-levels are fragmented into many small indexes. In contrast, the separate index approach integrates new indexes easily since all the individual indexes are maintained independently. Then, we propose new query processing methods for the top-*k* SKR query. First, we propose a new query processing method based on the separate index approach, namely, *SeparateSKR*. Second, we present two methods by extending representative methods for the top-*k* spatial keyword query based on the hybrid index approach to the top-*k* SKR query.



Object ID	Spatial Attribute	Textual Attribute	Relational Attribute
o_1	(2, 3)	… Mexican food…	10
o_2	(3, 1)	… movie theater…	5
o_3	(5, 1)	… hospital…	6
o_4	(8, 5)	… gas station…hospital…	1
o_5	(2, 7)	… hospital…hospital…	2
o_6	(7, 6)	… department store…	5

Top-1 Query:
- Query location: (3,7)
- Query keywords: {hospital}
- Query value: 3
- User preference: (1/3, 1/3, 1/3)

Object ID	Spatial Proximity	Textual Relevancy	Relational Closeness	Total Score
o_1	0.7	0.0	0.3	0.333
o_2	0.5	0.0	0.8	0.433
o_3	0.6	0.5	0.7	0.6
o_4	0.4	0.5	0.8	0.567
o_5	0.9	1.0	0.9	0.933
o_6	0.7	0.0	0.8	0.5

Figure 5 An example top-*k* SKR query

Fourth, through extensive experiments on the top-k SKR query using real and synthetic data sets, we compare efficiency of the methods in terms of the query performance and storage. We show that SeparateSKR is more efficient than the extended hybrid index methods by up to 13.30 times for the top-k MULTI query and by up to 448.68 times for single-type queries. We also show that SeparateSKR requires less storage than the extended hybrid index methods by up to 2.99 times.

The paper is organized as follows. Section 2 explains the top-k spatial keyword query and the processing methods for the query as a preliminary. Section 3 defines the top-k MULTI query. Section 4 discusses scalability of processing approaches for the top-k MULTI query. Section 5 investigates the top-k SKR query as an instance of the top-k MULTI query and the associated query processing method *SeparateSKR* based on the separate index approach. Section 6 presents performance evaluation. Section 7 summarizes and concludes the paper.

2 Preliminaries

2.1 The top-k spatial keyword query

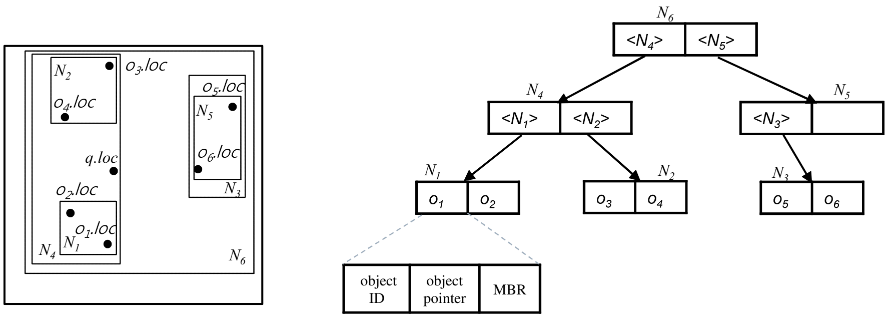
In the top-k spatial keyword query, each object o in the database D consists of 1) the spatial attribute $o.spatial$ and 2) the textual attribute $o.textual$. The query q consists of 1) the query location $q.location$, 2) the query keywords $q.keywords$, and 3) the user preference between the spatial attribute and the textual attribute $q.p$. The query returns k objects with the highest scores combining the component scores for the ranking measures. The ranking measures used are 1) spatial proximity and 2) textual relevance. Given a query q, the scoring function for an object o is as shown in (1) [8].

$$\begin{aligned} S(q,o) &= q.p \times S_{spatial}(q.location, o.spatial) \\ &\quad + (1-q.p) \times S_{textual}(q.keywords, o.textual) \end{aligned} \tag{1}$$

In (1), $S_{spatial}$ $(q.location, o.spatial)$ is a function for measuring spatial proximity between q and o; here, we use the Euclidean distance. Here, $S_{Spatial}(q.location, o.spatial)$ is normalized by the largest possible $S_{Spatial}(q.location, o.spatial)$ for a given data set. $S_{textual}(q.keywords, o.textual)$ is a function for measuring textual relevance between q and o; here, we use TF-IDF as shown in (2) [1]. TF $(o.textual, q.keywords_i)$ is the term frequency for $q.keywords_i$ in $o.textual$; DF $(D, q.keywords_i)$ the document frequency for $q.keywords_i$ in D. Here, $S_{Textual}(q.keywords, o.textual)$ is normalized by the largest possible $S_{Textual}(q.keywords, o.textual)$ for a given data set.

$$\begin{aligned} S_{textual}(q.keywords, o.textual) &= \sum_{i=1}^{n} S_{textual}(q.keywords_i, o.doc) \\ &= \sum_{i=1}^{n} TF(o.textual, q.keywords_i)\, log \frac{N}{DF(D, q.keywords_i)} \end{aligned} \tag{2}$$

We explain the spatial index and the textual index that can be used for efficient top-k spatial keyword query processing. Those indexes maintain the information that is needed to calculate the component scores for respective ranking measures (i.e., spatial proximity and



(a) Representation of objects and MBRs. (b) The structure of the spatial index for (a).

Figure 6 The spatial index

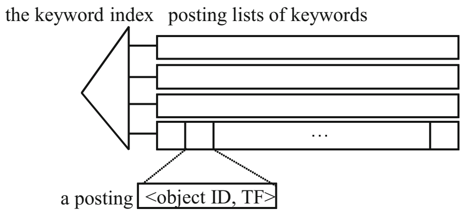
textual relevance). Figure 6 shows the spatial index. It is a hierarchical tree structure consisting of leaf nodes and internal nodes. An entry of the leaf node consists of the object ID, object pointer, and MBR of the object. The spatial index clusters the objects into leaf nodes and clusters the leaf nodes into internal nodes by their similarity. The representative spatial indexes are the R-tree family [2, 5, 13] and the MLGF family [14–16]. The difference between them is that the R-tree clusters the objects in the original space while the MLGF clusters them in the transform space. Figure 7 shows the inverted index. The inverted index is a representative textual index. It consists of keywords and their posting lists where each keyword has a posting list. Each keyword $keyword$ has $DF(D, keyword)$, and each posting for an object o in the posting list has <object ID, $TF(o.textual, keyword)$ >. For efficient searching for the keywords, a keyword index is used.

The existing methods for the top-k spatial keyword query are classified into two approaches: 1) the hybrid index approach and 2) the separate index approach. We explain the former in Section 2.2 and the latter in Section 2.3.

2.2 The hybrid index approach

There are two research directions in the hybrid index approach: 1) spatial-then-textual [3, 10] and 2) textual-then-spatial [12, 20, 21]. The former builds a spatial index on the entire set of objects based on the spatial attribute, partitioning it into groups (i.e., leaf nodes), and then, builds a textual index on the set of objects in each group. The latter builds a textual index on the entire set of objects, partitioning it into groups (i.e., posting lists), and then, builds a spatial index on the set of objects in each group. The representative method for the former is IR-tree [3, 10] and that for the latter is S2I [12].

Figure 7 The inverted index



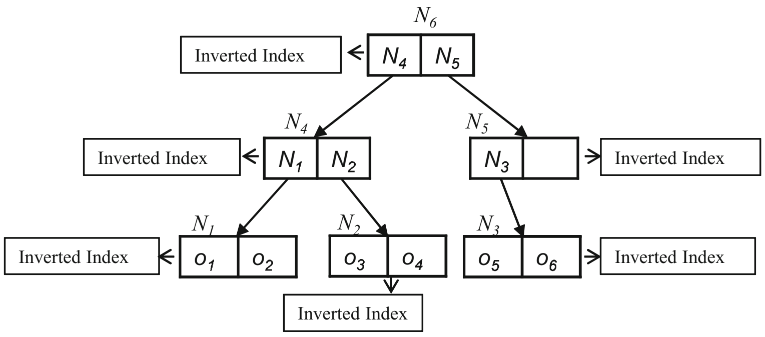


Figure 8 The index structure of IR-tree

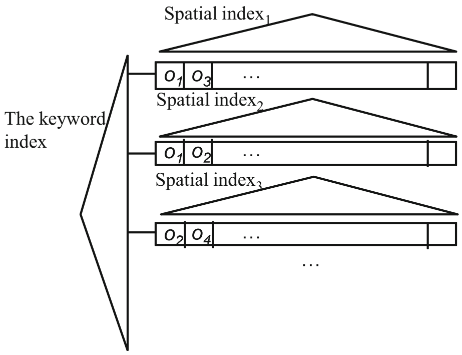
Figure 8 shows the index structure of IR-tree. IR-tree first builds a spatial index, and then, builds an inverted index on each leaf node and internal node of the spatial index. Hence, each node of the spatial index maintains not only spatial information but also textual information for each entry. IR-tree finds the top-*k* results by extending Incremental NN [6], which is a representative algorithm for finding *k* nearest neighbors based on the distance, so as to use the combined score of spatial proximity and textual relevance instead of the distance.

Figure 9 shows the index structure of S2I. S2I first builds an inverted index, and then, builds a spatial index on each posting list of the inverted index. Hence, each keyword maintains a spatial index that involves only those objects containing the keyword itself. S2I finds the top-*k* results by extending the query processing method of IR-tree to multiple indexes for a given set of query keywords. We note that S2I is desirable when the number of query keywords is small since we need to consider only those objects that contain the keywords. However, since each object contains multiple keywords, the same object is stored in multiple indexes redundantly. Thus, when the number of query keywords is large, S2I is not desirable.

2.3 The separate index approach

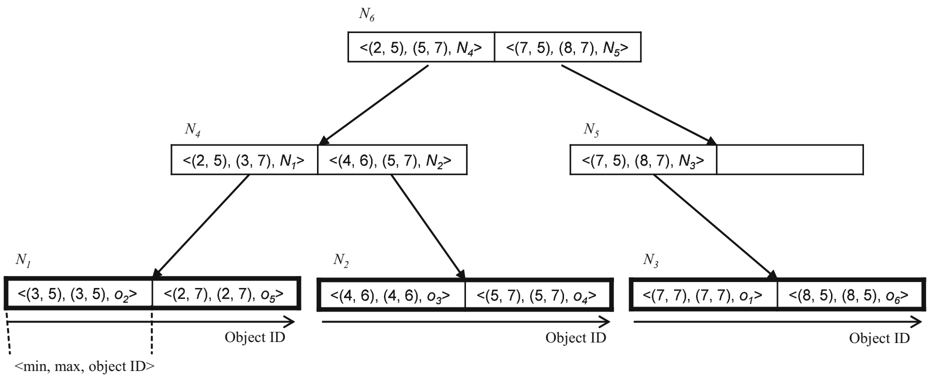
In the separate index approach, we need to support top-*k* pruning while supporting efficient merging of objects that are retrieved from the spatial and textual indexes. However, it has been considered difficult to support both top-*k* pruning and efficient merging since

Figure 9 The index structure of S2I

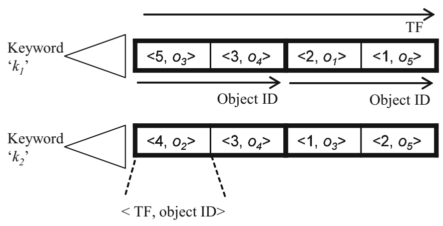


either one requires a different criterion for clustering the objects: 1) the score for the former and 2) the object ID for the latter. The rank-aware separate index method (RASIM) is the method based on the separate index approach for the top-*k* spatial keyword query [8]. RASIM supports clustering of the objects by two different criteria by employing the partitioning technique. That is, it partitions the entire set of objects into a set of groups sorted by the score for top-*k* pruning and sorts the objects in each group by the object ID for efficient merging. Here, we use a disk page as a group for controlling physical access.

Figure 10 shows the index structure of RASIM. It consists of a *rank-aware spatial index* for the spatial index and a *rank-aware inverted index* for the textual index. Figure 10a shows a rank-aware spatial index. Each entry of the leaf node consists of $< min(x, y), max(x, y), objectID >$. Here, $min(x, y)$ and $max(x, y)$ represent the MBR of the object. The rank-aware spatial index partitions the entire set of objects into leaf nodes and sorts the objects in a leaf node in the order of the object ID. Here, a leaf node is mapped to a disk page. Figure 10b shows a rank-aware inverted index. It partitions the entire set of objects into posting lists and further partitions the set of objects in each posting list in the unit of a disk page according to textual relevance (i.e., TF). We sort the objects in a group in the order of the object ID. When the query is given, RASIM retrieves a ranked list of groups in the order of the score from the rank-aware spatial index and the rank-aware inverted index, respectively. It applies the threshold algorithm (TA) [4] to the two ranked lists in the unit of groups to support top-*k* pruning while merging objects in the groups based on the object ID.



(a) Rank-aware spatial index.



(b) Rank-aware inverted index.

Figure 10 The index structure of RASIM

Zhang et al. [22] have proposed the method that models the top-k SK query as $m + 1$ sorted lists and that uses the TA algorithm, which is similar to RASIM [8]. The biggest difference between Zhang et al.'s method and RASIM is the spatial index structure used. The former uses the space-filling curve (i.e., Z-order) while the latter uses the multidimensional index (i.e., MLGF). In this paper, we only consider the clustering criterion (i.e., relational, spatial and textual attributes) without considering the specific index structures used to implement them. The multidimensional index structure can be used for not only the separate index approach but also the hybrid index approach, but the space-filling curve cannot be used for the hybrid index approach. For the sake of fairness, in Section 5, we use the multidimensional index structure for both the hybrid index approach and the separate index approach when we devise the methods for the top-k SKR query.

3 The top-*k* multiple-type integrated (MULTI) query

3.1 Problem definition

In this section, we define the *top-k MULTI query* formally. The top-k MULTI query is a generalized form of the top-k spatial keyword query. That is, while the top-k spatial keyword query deals with only two data types, i.e., the spatial data type and the textual data type, the top-k MULTI query deals with multiple arbitrary data types including the spatial, textual, or relational data types. In general, each data type can have multiple attributes. Nevertheless, without loss of generality, we assume we have only one attribute for each data type to focus on different data types. We then assign a ranking measure to each attribute. The final score of an object is calculated by summing up the component scores weighted by the user preference as shown in (3). Here, $S_i(q, o)$ is a function to calculate the component score for an attribute i, and $q.p_i$ is the user preference for i where $q.p_i (1 \leq i \leq d) \geq 0$. We normalize $q.p_i$'s so that $\sum_{i=1}^{d} q.p_i = 1$. Here, $S_i(q, o)$ is normalized by the largest possible $S_i(q, o)$ for a given data set.

$$S_{MULTI}(q, o) = \sum_{i=1}^{d} S_i(q, o) \times q.p_i \tag{3}$$

We define a new specific instance of the top-k MULTI query: the *top-k spatial-keyword-relational query* (simply, the *top-k SKR query*). The top-k SKR query is a query extended from the top-k spatial keyword query by adding the relational data type. In a top-k SKR query, the object o consists of 1) a spatial attribute $o.spatial$, 2) a textual attribute $o.textual$, and 3) a relational attribute $o.relational$. The top-k SKR query consists of 1) a query location $q.location$, 2) a query keywords $q.keywords$, and 3) a relational query value $q.relational$. The top-k SKR query uses a ranking measure for each attribute: 1) spatial proximity, 2) textual relevance, and 3) relational closeness.

We calculate relational closeness between q and o as shown in (4). Here, $o.relational$ is the relational value of the object, and $q.relational$ is the relational value of the query. If the relational data type is not numeric (i.e., date, time, char, or varchar), we first convert the attribute values to the numeric values following a linear order. Here, $S_{relational}(q, o)$ is normalized by the large possible $S_{relational}(q, o)$ for a given data set.

$$S_{relational}(q, o) = |q.relational - o.relational| \tag{4}$$

In this paper, we focus on the efficiency of query processing instead of the effectiveness of a ranking function. The query processing method we propose is valid for any ranking function as long as it is monotone with respect to each component score $S_i(q, o)$.

3.2 Comparison with the top-k query

In this section, we clarify the difference between the top-k MULTI query and the top-k query [7]. The top-k query returns k objects with the highest scores according to the scoring function as shown in (5) [7].

$$S_{TOPK}(o) = \sum_{i=1}^{d} S_i(o) \times q.p_i \tag{5}$$

The main difference of the top-k MULTI query and the top-k query relies on the scoring function $S_i()$ for each component score as shown in (3) and (5). The former calculates the component score from the given query and the attribute value of each object; the latter uses the attribute value itself as the component score. That is, the top-k MULTI query deals with the component scores that are determined dependent of the query, i.e., $S_i(q, o)$, while the top-k query deals with the component scores that are determined independent of the query, i.e., $S_i(o)$.

4 Scalability of processing methods for the top-k multiple-type integrated query

We consider the hybrid index and separate index approaches in the top-k spatial keyword query as the possible candidate approaches for the top-k MULTI query. In this section, we discuss the scalability of these approaches as more data types are newly integrated.

4.1 Scalability of the hybrid index approach

In the hybrid index approach, we need to build all the indexes for the data types involved in the top-k MULTI query in an integrated form. Figure 11 shows a general form of the hybrid index for processing the top-k MULTI query when N data types are involved. In the hybrid index approach, we build all the indexes involved in multiple levels. We call the index for level i the ith-level index. We build the 1st-level index for the entire set of objects, partitioning the objects into groups. Then, we build the 2nd-level index for the objects in each group, partitioning the group into sub-groups. We repeat this process until the Nth-level index is built.

We can model the hybrid index as a combination of multiple indexing and partitioning. Hence, we partition the objects into groups. We then build an index on the groups based on one data type and an index on the objects in each group based on the other data type. When N data types are involved, we formulate a hybrid index as shown in (6). In $indexing(A) \rightarrow indexing(B)$, $\rightarrow$ means partitioning the objects into groups based on A; $indexing(A)$ building an index on the groups based on A; $indexing(B)$ building an index on the objects for each group based on B. We can obtain a possible hybrid index method per each permutation of the attributes as in (6). As a result, we obtain $N!$ different methods.

$$indexing(Attribute_1) \rightarrow indexing(Attribute_2) \rightarrow ... \rightarrow indexing(Attribute_N) \tag{6}$$

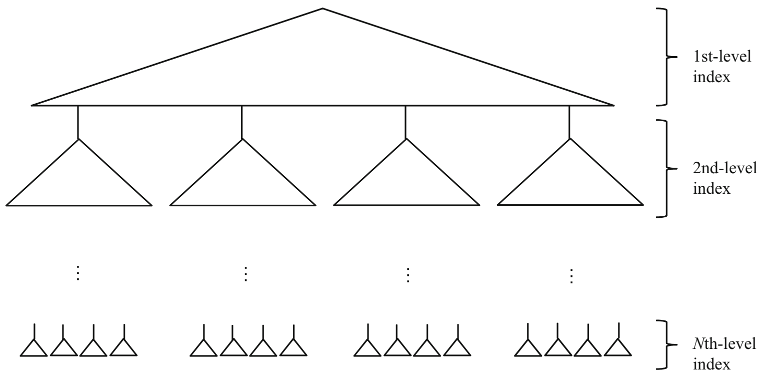


Figure 11 A general form of the hybrid index for processing the top-k MULTI query

Let us consider the top-k spatial keyword query as an instance of the top-k MULTI query. According to (6), we can generate two methods: 1) $indexing\ (SpatialAttribute) \rightarrow indexing\ (TextualAttribute)$ and 2) $indexing(TextualAttribute) \rightarrow indexing\ (SpatialAttribute)$. The former is mapped to IR-tree [3] explained in Section 2; the latter to S2I [12].

The hybrid index methods have the following two limitations in terms of scalability as new data types are integrated. 1) The indexes are excessively fragmented; the lower the index level is, the more severe the fragmentation is. For example, while IR-tree builds one spatial index on the entire set of objects, it builds multiple fragmented inverted indexes for the leaf nodes of the spatial index. 2) The hybrid index methods are inefficient for processing partial-types of the top-k MULTI query, e.g., only spatial queries or only keyword queries since they inherently require accesses to the part of the index for the other data types despite that they are not necessary.

4.2 Scalability of the separate index approach

Figure 12 shows a general form of the separate index for processing the top-k MULTI query when N data types are involved. Unlike in the hybrid index approach, in the separate index approach, new indexes can be easily added since each individual index is maintained independently.

Let us extend RASIM [8] to deal with additional data types. For building the index structures, we only add new indexes while maintaining the existing indexes (i.e., the rank-aware

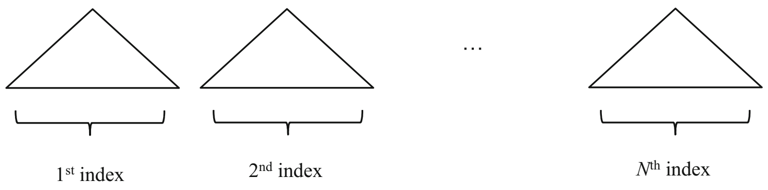


Figure 12 A general form of the separate index for processing the top-k MULTI query

spatial index and the rank-aware spatial index) unchanged. For processing queries, since RASIM uses the threshold algorithm (TA) [4], which is originally designed to deal with multiple lists, we can naturally extend it by adding lists retrieved from the indexes for the new data types. In Section 5, we will propose a specific method based on the separate index approach for the top-*k* SKR query, a specific instance of the top-*k* MULTI query.

5 Top-*k* spatial-keyword-relational (top-*k* SKR) query processing

In this section, we discuss top-*k* SKR query processing. In Section 5.1, we investigate the hybrid index methods for the top-*k* SKR query. In Section 5.2, we propose a new separate index method, *SeparateSKR*, for the top-*k* SKR query.

5.1 The hybrid index methods

If we apply the relational, spatial, and textual attributes into (6) in Section 4.1, we obtain the six methods for the top-*k* SKR query in Table 1. Every method in Table 1 builds the relational, spatial, and textual indexes at three levels. For example, in Method (1), we first build a relational index on the entire set of the objects, partitioning them into groups. Then, we build a spatial index on the objects in each group, partitioning them into sub-groups. Finally, we build a textual index on the objects in each sub-group.

We mainly examine the two methods (4) and (6), which are extensions of IR-tree [3] and S2I [12]. That is, Method (4) integrates the relational index with IR-tree; Method (6) integrates it with S2I. Methods (4) and (6) build the relational index on each sub-group partitioned by the 2nd-level index (i.e., the textual index for IR-tree and the spatial index for S2I). We name the former *IRtree-Relational* and the latter *S2I-Relational*. Figure 13 shows the index structure of IRtree-Relational. For IRtree-Relational, we first build a spatial index on the entire set of objects. Then, we build an inverted index for each leaf node of the spatial index. Last, we build a relational index on each posting list of the inverted index. Figure 14 shows the index structure of S2I-Relational. For S2I-Relational, we first build an inverted on the entire set of objects. Then, we build a spatial index for each posting list of the inverted index. Last, we build a relational index on each leaf node of the spatial index.

We revisit the two limitations of IRtree-Relational and S2I-Relational. 1) They make many fragmented relational indexes. This means that they cannot effectively support clustering for the relational attribute. 2) They are not efficient in processing partial-type integrated queries. In particular, the single-type query (i.e., the spatial-only query, keyword-only query, or relational-only query) processing is extremely inefficient.

Table 1 The hybrid index methods

(1) $indexing(RelationalAttribute) \rightarrow indexing(SpatialAttribute) \rightarrow indexing(TextualAttribute)$
(2) $indexing(RelationalAttribute) \rightarrow indexing(TextualAttribute) \rightarrow indexing(SpatialAttribute)$
(3) $indexing(SpatialAttribute) \rightarrow indexing(RelationalAttribute) \rightarrow indexing(TextualAttribute)$
(4) $indexing(SpatialAttribute) \rightarrow indexing(TextualAttribute) \rightarrow indexing(RelationalAttribute)$
(5) $indexing(TextualAttribute) \rightarrow indexing(RelationalAttribute) \rightarrow indexing(SpatialAttribute)$
(6) $indexing(TextualAttribute) \rightarrow indexing(SpatialAttribute) \rightarrow indexing(RelationalAttribute)$

5.2 SeparateSKR: a separate index method for the top-*k* SKR query

5.2.1 Index structure

For SeparateSKR, we build the index for each attribute separately. We use the primitive index structures (e.g., R-tree, inverted index, and B+-tree) themselves but modifying only their data clustering. Index structures for SeparateSKR consist of 1) the spatial index, 2) the textual index, and 3) the relational index. We only need to add the relational index while using the existing separate index method for the top-*k* spatial keyword query. Thus, for the spatial index and the textual index, we can use the same indexes as RASIM [8], i.e., the rank-aware spatial index for the former and the rank-aware inverted index for the latter. For the relational index, we propose a new index structure, *rank-aware B+-tree*, based on the B+-tree. The rank-aware B+-tree partitions the entire set of objects based on the relational attribute into the leaf nodes and sorts the objects in each leaf node in the order of the object ID. Figure 15 shows the rank-aware B+-tree. Here, we also map a leaf node to a disk page. In Section 5.2.2, we will explain how we access the leaf nodes in the order of relational closeness using the rank-aware B+-tree.

5.2.2 Query processing algorithm

We can easily extend the query processing algorithm of RASIM [8], the separate index method for the top-*k* spatial keyword query, to a query processing algorithm for the top-*k* SKR query by adding a list retrieved from the relational index.

We first explain the query processing algorithm of RASIM. It extends the threshold algorithm (TA) [4] to deal with the sorted list of groups instead of the sorted list of individual objects. That is, it retrieves the objects in the unit of group from each attribute and examines the objects in a group at the same time. It repeats the following three steps until the top-*k*

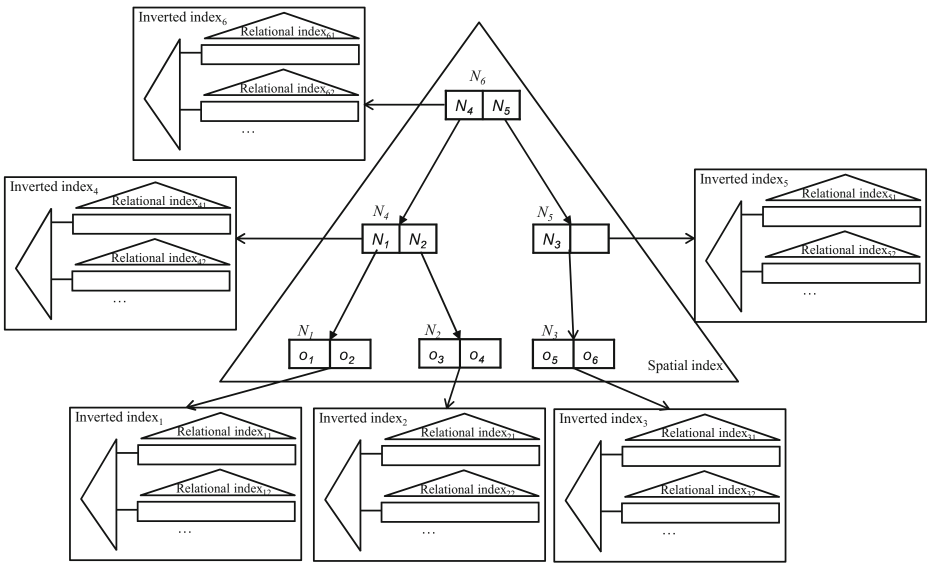


Figure 13 The index structure of IRtree-Relational

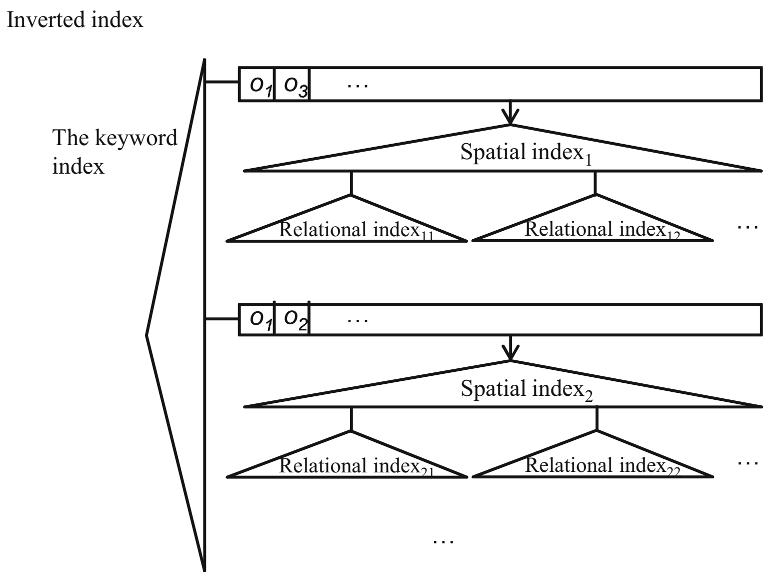


Figure 14 The index structure of S2I-Relational

results are found: 1) the retrieving step, 2) merging step, and 3) scoring step. In the retrieving step, we retrieve the most relevant group from each of the spatial or textual attributes. The *most relevant group* is determined as the group containing the object with the highest component score among the groups that have not yet been retrieved. In the merging step, we merge the objects in the most relevant groups retrieved from each attribute using the object ID as the merging attribute. In the scoring step, we calculate the scores of the objects in the merged results and retrieve the top-*k* results. Here, if the object is included in the most relevant group only from one attribute, we obtain its component score for the other attribute by directly accessing the attribute value as in the threshold algorithm. We calculate the *threshold*, which is the highest possible score of the objects that have not yet been retrieved, by substituting the highest component score of the objects that have been included in the most

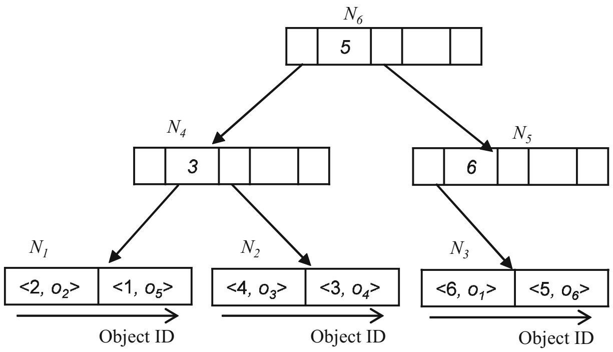


Figure 15 The structure of the rank-aware B+-tree

```
Algorithm IncrementalRC
Input: 1) root: the root of the rank-aware B+-tree
       2) qv: relational query value
Output: MRGroup: the most relevant group

RC := relational closeness between qv and root;
PQ.push (root, RC);        /* PQ is a priority queue to store the objects
                              in the order of relational closeness */
WHILE (PQ is not empty)
  Element = PQ.pop();
  IF Element is the internal node THEN
    FOR (i = 1; i <=nEntry; i++)
      iNode = Element.ChildNode(i);    /* i th child node of Element */
      RC := relational closeness between qv and iNode;
      PQ.push(iNode, RC);   /* Nodes are sorted in the order of relational closeness */
    END
  ELSE IF Element is a leaf node THEN
      MRGroup =Element;
      break;
  END
END
RETURN MRGroup;
```

Figure 16 The Incremental RC algorithm

relevant group for each attribute into the scoring function. Thus, we retrieve k objects whose scores are higher than the threshold in the order of the score as the results.

The only difference between SeparateSKR and RASIM is that the former requires an additional index for the relational attribute. Thus, we propose a method, *Incremental Relational Closeness (Incremental RC)*, to retrieve the most relevant group from the rank-aware B+-tree. We modify Incremental NN [6], which has been proposed for the R-tree, to adapt it to the rank-aware B+-tree so as to retrieve the groups in the order of relational closeness. Figure 16 shows Incremental RC. Here, to calculate relational closeness between the relational query value and each node, we use the relational value range of the objects stored in the node. If the query value is included in the value range of the node, relational closeness between the query and the node is 0, which means the closest. Otherwise, it is the minimum difference between the relational query value and the value range of the node.

6 Performance evaluation

6.1 Experimental data and environment

We compare the index size and query performance of SeparateSKR with those of the hybrid index methods extended for the top-k SKR query. For the extended hybrid index methods, we implement IRtree-Relational and S2I-Relational as presented in Section 5.1. For SeparateSKR, we implement the index structures and query processing algorithm as presented in Section 5.2. We use the MBR-MLGF [14], which is one of the MLGF family [15], for the spatial index, the inverted index for the textual index, and the B+-tree for the relational index. We implemented all the methods using the MBR-MLGF, inverted index, and B+-tree index that are part of the Odysseus DBMS[1] [17–19]. For the sake of the fairness, we

[1]Each entry of the index contains an object ID of 16 bytes.

Table 2 Characteristics of the data sets

Data sets	Real data sets		Synthetic data sets			
	DataSet1	*DataSet2*	*DataSet3*	*DataSet4*	*DataSet5*	*DataSet6*
Total number of objects	78,260	131,461	100,000	200,000	500,000	1,000,000
Maximum number of postings per keyword	55,949	131,461	88,320	152,260	354,175	794,915
Average number of postings per keyword	89	168	483	389	538	818
Total number of unique words in the data set	206,636	114,831	33,330	94,920	217,231	318,916
Total number of words in the data set	18,397,075	19,278,878	16,114,000	36,918,340	116,863,650	261,040,074

use the same index for all the methods. To compare the query performance, we measure the elapsed time and the number of page accesses.

For SeparateSKR, we can adjust the number of groups to access at a time from each index. We expect that a large accessing unit improves the effect of the sequential access, but may retrieve lots of unnecessary objects. Thus, when the desired number of results k is small (large), a small (large) accessing unit is preferred. In the experiment, we show the performance of SeparateSKR as the size of the accessing unit is varied.

We use six data sets. Table 2 shows the characteristics of the real and synthetic data sets. *DataSet1* consists of 1) the spatial attribute representing the buildings in Seoul, 2) the textual attribute representing Web pages crawled, and 3) the relational attribute representing the areas of the real estates in Seoul.[2] *DataSet2*[3] consists of 1) the spatial attribute representing locations in Los Angeles,[4] 2) the text attribute representing the texts in the 20Newsgroups dataset,[5] and 3) the relational attribute representing the prices of real estates in Los Angeles.[6] To measure the query performance as the data set size is varied, we generate four synthetic data sets of varying sizes: 1) *DataSet3* (100K), 2) *DataSet4* (200K), 3) *DataSet5* (500K), and 4) *DataSet6* (1M). Each of them consists of 1) the spatial attribute, 2) the textual attribute, and 3) the relational attribute, where all the attributes are randomly generated.

We generate five query sets for each data set, where the number of query keywords is 1, 2, 3, 4, and 5, respectively. Each query set consists of 100 queries where query locations and query values are randomly generated. Query keywords are randomly selected from a set of keywords whose document frequencies are greater than one percent of the total number of objects—effectively excluding infrequent or unrealistic keywords. We use the average query performance over 100 queries. Table 3 shows the parameters for the evaluation. The default parameter values are represented in bold. For the user preference, we focus on its variation only for the relational attribute. A default query used in the experiments consists

[2]http://www.molit.go.kr

[3]For *DataSet2*, we extend the data set used in the existing methods [3, 8, 10] for the top-k spatial keyword query by adding the relational attribute.

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

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

[6]http://www.laalmanac.com

Table 3 Query parameters and values

Parameters	Description	Values
k	The number of desired results	1, 5, **10**, 20, 50
$nKeywords$	The number of query keywords	1, 2, **3**, 4, 5
$p_i (1 <= i <= 3)$	User preference vector where p_1 is for the spatial attribute, p_2 for the textual attribute, and p_3 for the relational attribute	[0.45, 0.45, 0.1], [0.35, 0.35, 0.3], **[0.33, 0.33, 0.33]**, [0.25, 0.25, 0.5], [0.15, 0.15, 0.7], [0.05, 0.05, 0.9]
$nData$	Dataset size	**78K**, 100K, 200K, 500K, 1M

of a spatial constant, a relational constant, and three query keywords. This type of query is fairly complex, but there have been no earlier research results for the query type.

For the sake of evaluating a lower-bound (i.e., the worst-case) performance, we run all the methods at cold start. *Cold start* means an environment where the buffering effect of the LINUX file system is completely removed. To guarantee cold start, we use raw disks for storing data and indices. We conduct all the experiments on a Pentium4 3.6GHz Linux machine with 2.5GB of main memory. The page size for data and indexes is set to 4,096 bytes.

6.2 Results of the experiments

6.2.1 Index size

Table 4 shows the index size of each method for each data set. It shows that the index size of SeparateSKR is reduced by up to 1.79 times compared with IRtree-Relational and by up to 2.99 times compared with S2I-Relational. In IRtree-Relational, the inverted indexes and relational indexes reside redundantly in the internal nodes and the leaf nodes. In S2I-Relational, the spatial indexes and the relational indexes on the posting lists of the inverted index have redundant entries for the same object since an object is stored in multiple posting lists.

6.2.2 Query performance of SeparateSKR as the size of the accessing unit is varied

Figure 17 shows the query performance of SeparateSKR as the size of the accessing unit (i.e., the number of groups to access at a time) is varied from 4 groups to 32 groups. The result shows that, as k increases, the query performance of SeparateSKR with a large accessing unit becomes more efficient. For the rest of the experiments, we use 16 groups as the

Table 4 Sizes of indexes (MB)

Method	*DataSet1*	*DataSet2*	*DataSet3*	*DataSet4*	*DataSet5*	*DataSet6*
SeparateSKR	1,019	913	814	1,833	5,912	13,468
IRtree-Relational	1,223	1,634	1,029	2,840	9,865	22,745
S2I-Relational	3,043	2,323	1,220	3,200	9,070	20,270

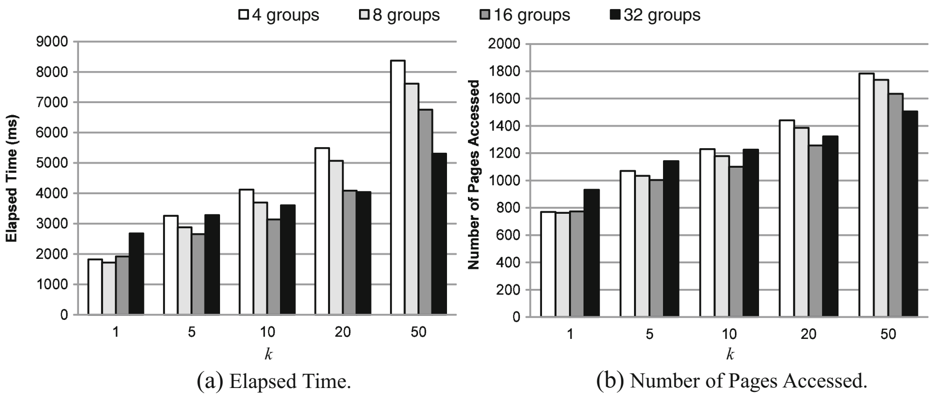


Figure 17 Query performance for *DataSet*1 as the size of the accessing unit of *SeparateSKR* is varied

accessing unit where SeparateSKR shows the best performance for $k = 10$, which is the default value of k.

6.2.3 *Query performance as k is varied*

Figure 18 shows the query performance of SeparateSKR and the extended hybrid index methods as k is varied. The result shows that SeparateSKR outperforms the extended hybrid index methods in elapsed time by $4.70 \sim 10.69$ times compared with IRtree-Relational and by $1.80 \sim 3.62$ times compared with S2I-Relational. The number of pages accessed shows a similar trend. This efficiency of SeparateSKR is from the fact that the extended hybrid index methods cannot effectively support clustering for relational closeness while SeparateSKR supports clustering for each attribute.

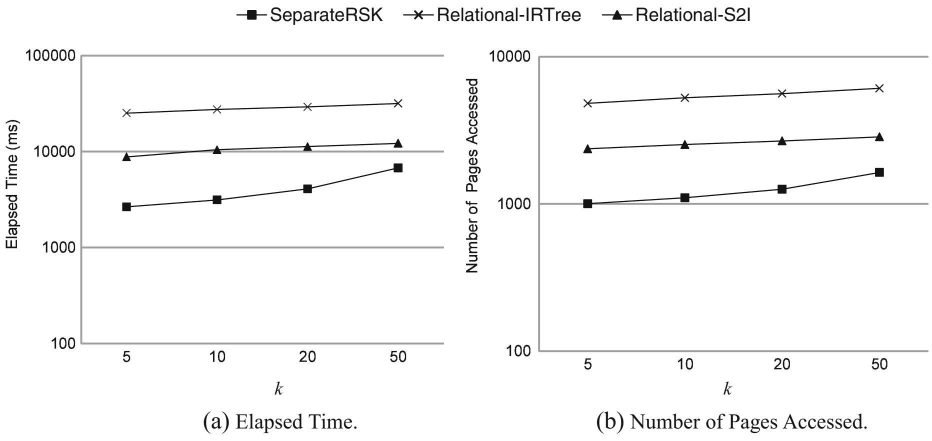


Figure 18 Query performance for *DataSet*1 as k is varied

6.2.4 Query performance as $nKeywords$ is varied

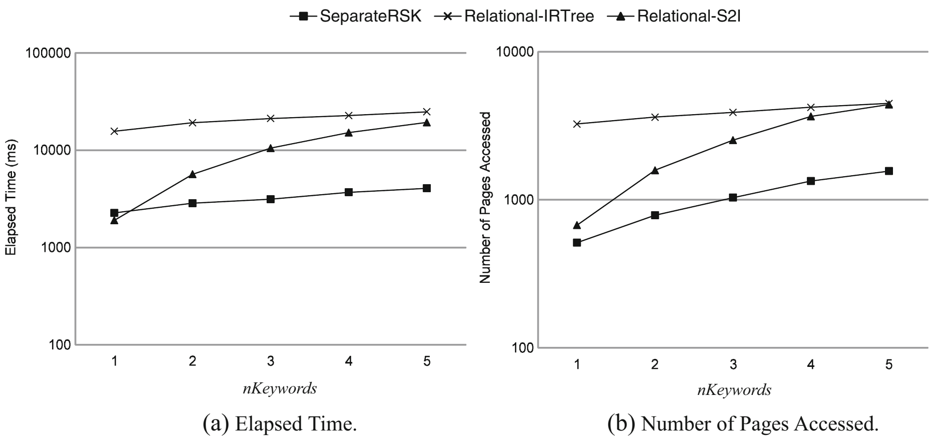
Figure 19 shows the query performance of SeparateSKR and the extended hybrid index methods as $nKeywords$ is varied. The result shows that SeparateSKR outperforms the extended hybrid index methods in elapsed time by 6.10 $\sim$ 6.90 times compared with IRtree-Relational and by 0.84 $\sim$ 4.75 times compared with S2I-Relational. The query performance of S2I-Relational is degraded much faster than the other methods as $nKeywords$ increases. Since S2I-Relational involves only those indexes for given query keywords, the number of indexes to access increases as the number of query keywords increases.

6.2.5 Query performance as the user preference for the relational attribute increases

Figure 20 shows the query performance of SeparateSKR and the extended hybrid index methods as the user preference for the relational attribute increases from 0.1 to 0.9. The result shows that the query performance of SeparateSKR becomes much more efficient than those of the extended hybrid index methods as the user preference for the relational attribute increases. This result stems from the fact that SeparateSKR takes advantage of clustering for the relational attribute while the extended hybrid index methods do not. It also indicates that performance degradation of IRtree-Relational is more severe than that of S2I-Relational since the former involves the entire index while the latter involves only those indexes for a set of given keywords in the query. As a result, the elapsed time of SeparateSKR is reduced by 1.87∼9.61 times, compared with that of IRtree-Relational and by 1.98∼3.26 times compared with that of S2I-Relational.

6.2.6 Query performance of single-type queries

Figure 21 shows the query performance of SeparateSKR and the extended hybrid index methods for the single-type queries. The result shows that SeparateSKR is much more



(a) Elapsed Time. (b) Number of Pages Accessed.

Figure 19 Query performance for $DataSet1$ as $nKeywords$ is varied

efficient than the hybrid index methods for the single-type queries. Especially, for the relational queries, the query performance of the extended hybrid index methods is degraded most since they build excessively many fragmented relational indexes. Specifically, compared with IRtree-Relational, the elapsed time of SeparateSKR is reduced by 2.92 times for the spatial-only query, 16.41 times for the keyword-only query, and 448.68 times for the relational-only query; compared with S2I-Relational, it is reduced by 1.50 times for the spatial-only query, 7.79 times for the keyword-only query, and 48.30 times for the relational-only query.

6.2.7 *Query performance as the dataset size is varied*

Figure 22 shows the query performance of SeparateSKR and the extended hybrid index methods as the dataset size is varied. The result shows that SeparateSKR is more efficient than the extended hybrid index methods constantly as the dataset size is varied. Specifically, the elapsed time of SeparateSKR is reduced by 4.34 $\sim$ 5.56 times compared with that of IRtree-Relational and by 1.82 $\sim$ 2.78 times compared with that of S2I-Relational.

6.2.8 *Query performance for* DataSet2 *as k is varied*

Figure 23 shows the query performance of SeparateSKR and the extended hybrid index methods on *DataSet2* as k is varied. The result shows that the query performance on *DataSet2* has a similar trend to that on *DataSet1*. Specifically, the elapsed time of SeparateSKR is reduced by 5.90 $\sim$ 13.30 times compared with that of IRtree-Relational and by 3.48 $\sim$ 5.73 times compared with that of S2I-Relational. We omit the experimental results on the other parameters for *DataSet2* since they also have trends similar to those for *DataSet1*.

6.2.9 *Query performance for* DataSet1 *as the number of data types increases*

Figure 24 compares the query performance of the top-k SKR query with that of the top-k spatial-keyword(SK) query to show the query performance as the number of data types

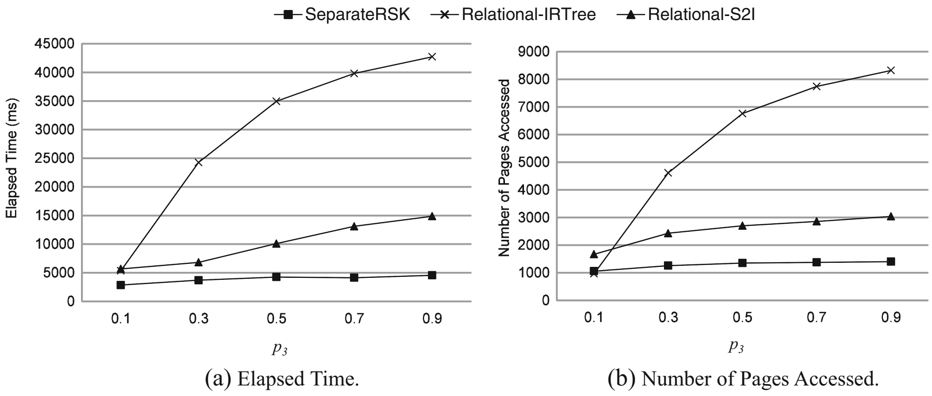


Figure 20 Query performance for *DataSet*1 as the user preference for the relational attribute p_3 increases

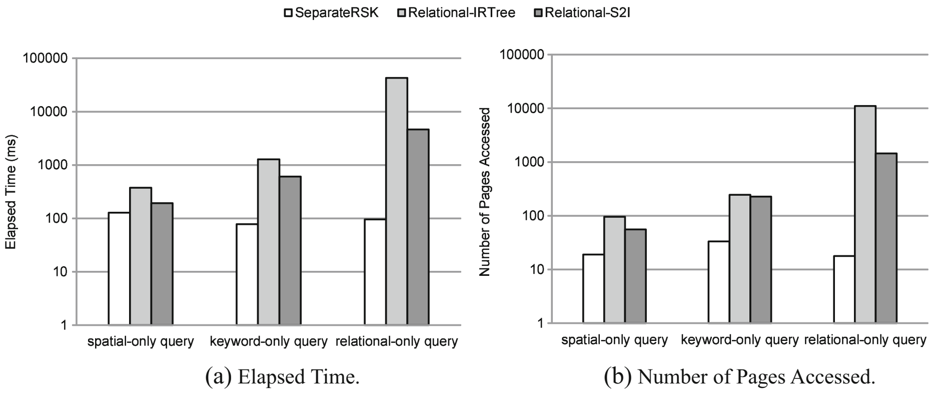


Figure 21 Query performance of single-type queries for *DataSet1*

increases. Let us define O_{Prune} as the number of objects to access until the top-k results are obtained. It is known that O_{Prune} is proportional to (7) as the number of data types increases [11]. Here, m is the number of data types; N data set size; k the number of results to retrieve.

$$m \times N^{\frac{(m-1)}{m}} \times k^{\frac{1}{m}} \tag{7}$$

Let us obtain O_{Prune} in the top-k SK query for a given data set, which we call $O_{Prune}(SK)$, and, O_{Prune} in the top-k SKR query for a given data set, which we call $O_{Prune}(SKR)$. Let us consider a query that obtains top-10 results for $DataSet1$. For the top-k SK query, the number of data types is 2 (i.e., spatial and keyword data types). For the top-k SKR query, the number of data types is 3 (i.e., spatial, keyword, and relational data types). By substituting them to the equation above, we get the following result. We get the following result: $O_{Prune}(SK) = 1769.29$, $O_{Prune}(SKR) = 11820.7$, $\frac{OPrune(SKR)}{OPrune(SK)} =$

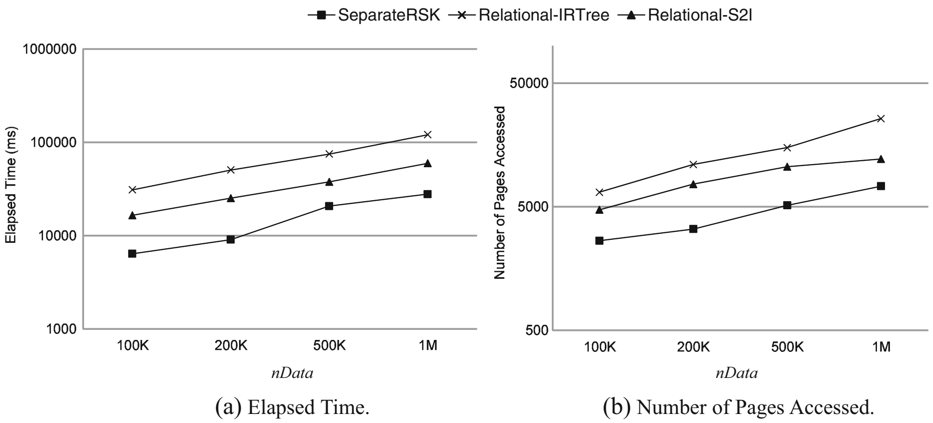


Figure 22 Query performance as the dataset size is varied

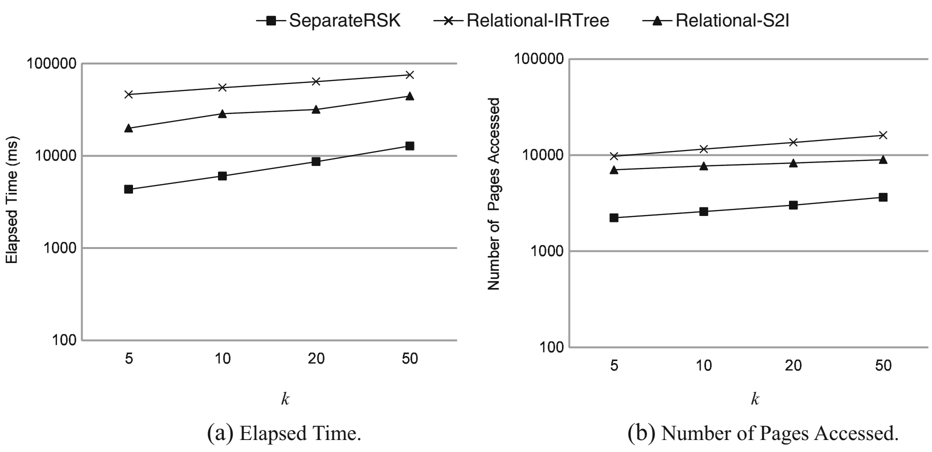


Figure 23 Query performance for $DataSet2$ as k is varied

6.69. This means that the query performance of the top-k SKR query could be less efficient than that of the top-k spatial keyword (SK) query about 6.69 times roughly.

To compare the query performance of the methods for the top-k SKR query with that of the methods for the top-k SK query, we measure the query performance of RASIM, IR-tree, and S2I for the top-k SK query and SeparateSKR, Relational-IRTree, and Relational-S2I for the top-k SKR query under the same experimental environment. Figure 24 shows their results. It indicates that SeparateSKR is less efficient than RASIM by 2.18 times in the elapsed time and by 2.86 times in the number of pages accessed; Relational-IRTree is less efficient than IR-tree by 12.16 times in the elapsed time and by 14.68 times in the number of pages accessed; Relational-S2I is less efficient than S2I by 3.61 times in the elapsed time and by 6.37 times in the number of pages accessed. Those results of the methods for the top-k SKR are quite reasonable if we consider that $\frac{OPrune(SKR)}{OPrune(SK)}$ is 6.69.We also know that the performance degradation of SeparateSKR compared with that of RASIM is less than the other methods.

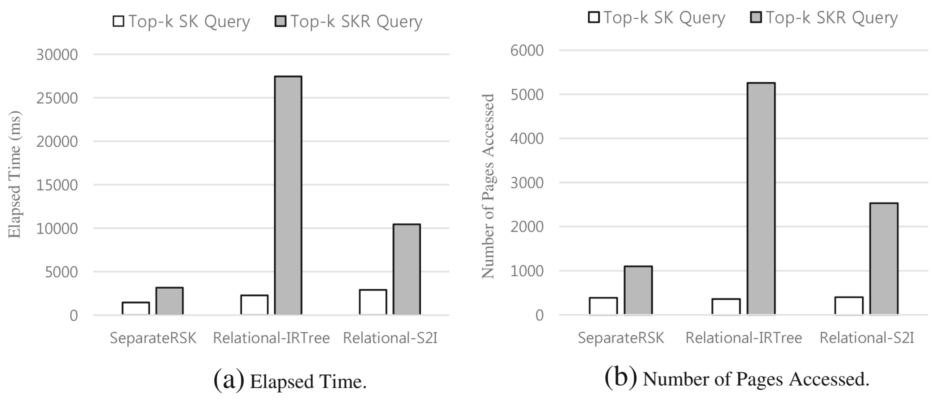


Figure 24 Query performance for $DataSet1$ as the number of data types increases

7 Conclusions

We have defined a new class of queries, the *Top-k MULTI query*. The top-*k* MULTI query deals with arbitrary multiple data types and returns the top-*k* results based on the relevance between the query and the object. The top-*k* MULTI query is practically important since we can find relevant results even if we do not know the exact values to find. The top-*k* spatial keyword query is an existing representative instance of the top-*k* MULTI query. It deals with the spatial data type and text data type and finds the information based on spatial proximity and textual relevance between the query and the object.

First, we have defined the top-*k* MULTI query formally. The top-*k* MULTI query distinguishes itself from the traditional top-*k* query in that the component scores to calculate final scores are determined dependent of the query. We have also defined a new specific instance of the top-*k* MULTI query, *top-k spatial-keyword-relational(SKR) query*, by integrating the relational data type into the top-*k* spatial keyword query.

Second, we have investigated the processing approaches for the top-*k* MULTI query based on hybrid index and separate index approaches. As a result, we have shown that the separate index approach is more scalable than the hybrid index approach as the new data types are integrated. We have also devised new processing methods for the top-*k* SKR query. First, we have proposed a new separate index method, *SeparateSKR*, for the top-*k* SKR query. Second, we have presented two methods by extending representative methods for the top-*k* spatial keyword query based on the hybrid index approach to the top-*k* SKR query.

Third, through extensive experiments, we have shown that SeparateSKR is more efficient than the extended hybrid index methods in terms of the query performance and the index size. Specifically, we have shown that SeparateSKR is more efficient than the extended hybrid index methods by up to 13.30 times for the top-*k* MULTI query and by up to 448.68 times for single-type queries. We have also shown that SeparateSKR requires less storage than the extended hybrid index methods by up to 2.99 times.

The top-*k* MULTI query are a general class of queries dealing with multiple ranking measures that are *query dependent*. As a new practical instance for the top-*k* MULTI query, we have dealt with the top-*k* SKR query. As more new data types are emerging, the other data types can also be integrated to the top-*k* MULTI query. Since the separate index method is scalable, it can be easily extended to accommodate a new data type for the top-*k* MULTI query.

Acknowledgments This work was supported by the National Research Foundation of Korea (NRF) grant funded by Korean Government.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. ACM Press, Addison-Wesley (1999)
2. Beckmann, N., Kriegel, H., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. In: Proceedings of the International Conference on Management of Data, ACM SIGMOD, pp. 322–331 (1990)
3. Cong, G., Jensen, C., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. In: Proceedings of the 35th International Conference on Very Large Data Bases (VLDB), pp. 754–765 (2009)
4. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), pp. 102–113 (2001)

5. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proceedings of the International Conference on Management of Data, ACM SIGMOD, pp. 47–57 (1984)
6. Hjaltason, G., Samet, H.: Distance browsing in spatial databases. ACM Trans. Database Syst. **24**(2), 265–318 (1999)
7. Ilyas, I., Beskales, G., Soliman, M.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. **40**(4), Article 11 (2008)
8. Kwon, H., Whang, K., Song, I., Wang, H.: RASIM: a rank-aware separate index method for answering top-k spatial keyword queries. World Wide Web J. **16**(2), 111–139 (2013)
9. Kwon, H., Wang, H., Whang, K.: G-index model: a generic model of index schemes for top-k spatial-keyword queries. World Wide Web J. doi:10.1007/s11280-014-0294-0 (2014)
10. 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)
11. Pang, H., Ding, X., Zheng, B.: Efficient processing of exact top-k queries over disk-resident sorted lists. VLDB J. **19**(3), 437–456 (2010)
12. Rocha-Junior, J., Gkorgkas, O., Jonassen, S., Norvag, K.: Efficient processing of top-k spatial keyword queries. In: Proceedings 12th International Symposium on Spatial and Temporal Databases (SSTD), pp. 205–222 (2011)
13. Sellis, T., Roussopoulos, N., Faloutsos, C.: The R+-tree: a dynamic index for multi-dimensional objects. In: Proceedings of the 13th International Conference on Very Large Data Bases (VLDB), pp. 507–518 (1987)
14. Song, J., Whang, K., Lee, Y., Kim, S.: Spatial join processing using corner transformation. IEEE Trans. Knowl. Data Eng. **11**(4), 688–698 (1999)
15. Whang, K., Krishnamurthy, R.: The multilevel grid file: a dynamic hierarchical multidimensional file structure. In: Proceedings of the International Symposium on Database Systems for Advanced Applications (DASFAA), pp. 449–459 (1991)
16. Whang, K., Kim, S., Wiederhold, G.: Dynamic maintenance of data distribution for selectivity estimation. VLDB J. **3**(1), 29–51 (1994)
17. Whang, K., Lee, M., Lee, J., Kim, M., Han, W.: Odysseus: a high-performance ORDBMS tightly-coupled with IR features. In: Proceedings of the 21st International Conference on Data Engineering (ICDE), IEEE, pp. 1104–1105. This paper received the Best Demonstration Award (2005)
18. 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)
19. Whang, K., Lee, J., Han, W., Kim, M., Kim, J.: DB-IR integration using tight-coupling in the Odysseus DBMS. World Wide Web J., to appear (2013)
20. Zhang, D., Tan, K., Tung, A.: Scalable top-k spatial keyword search. In: Proceedings of the 16th International Conference on Extending Database Technology (EDBT), pp. 359–370. ACM (2013)
21. Zhang, C. et al.: Inverted linear quadtree: efficient top-k spatial keyword search. In: Proceedings of the 29th International Conference on Data Engineering (ICDE), IEEE, pp. 901–9012 (2013)
22. Zhang, D., Chan, C., Tan, K.: Processing spatial keyword query as a top-k aggregation query. In: Proceedings 37th International Conference on Research and Development in Information Retrieval, ACM SIGIR, pp. 355–364 (2014)