World Wide Web (2015) 18:969–995
DOI 10.1007/s11280-014-0294-0

G-Index Model: A generic model of index schemes for top-k spatial-keyword queries

Hyuk-Yoon Kwon · Haixun Wang · Kyu-Young Whang

Received: 26 October 2013 / Revised: 27 January 2014 /
Accepted: 12 May 2014 / Published online: 13 June 2014

Abstract A top-k spatial-keyword query returns the k best spatio-textual objects ranked based on their proximity to the query location and relevance to the query keywords. Various index schemes have been proposed for top-k spatial-keyword queries; however, a unified framework covering all these schemes has not been proposed. In this paper, we present a generic model of index schemes for top-k spatial-keyword queries, which we call *G-Index Model*. First, G-Index Model is a unified framework that exhaustively investigates all the possible index schemes for top-k spatial-keyword queries. For this, we conjecture that data clustering is the key element in composing various index schemes and generate index schemes as combinations of clustering. The result shows that all the existing methods map to those generated by G-Index Model. Using G-Index Model, we also discover two new methods that have not been reported before. Second, we show that G-Index Model is generic, i.e., it can generate index schemes for a class of queries integrating arbitrary multiple data types. For this, we show that G-Index Model can enumerate index schemes for two classes of queries: the spatial-keyword query (without the top-k constraint) and the top-k spatial-keyword-relational query, which adds the relational data type to the top-k spatial-keyword query. Third, we propose a cost model of the generated methods for the top-k spatial-keyword query. Consequently, the cost model allows us to do physical database design so as to find an optimal index scheme for a given usage pattern (i.e., a set of query loads and frequencies). We validate the cost model through extensive experiments.

Keywords Top-k spatial-keyword query · Generic model · Cost model

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

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

H. Wang
Google Research, Mountain View, CA, USA
e-mail: haixun@google.com

1 Introduction

With the ubiquity of mobile devices, location data is ever abundant. One study found that about one fifth of Web searches are geographical, that is, the searches have location intent [18]. More and more people have been using Google Maps, online yellow pages, and other location-based services. The representative queries in these applications are the top-k spatial-keyword queries. A top-k spatial-keyword query returns the k best spatio-textual objects ranked based on their proximity to the query location and relevance to the query keywords [7]. In Figure 1, we show six objects $o_1, \cdots, o_6$ in a geographical space, where each object has a textual label. A top-k spatial-keyword query q is also shown in the figure. Query q may be interpreted as: *Find top-k objects whose locations are the closest to q's location and whose textual labels are the most relevant to keywords 'vegetable' and 'food'.* Assuming $k = 2$, the query retrieves o_1 and o_5 based on spatial proximity as well as textual relevancy.

Several index schemes have been proposed for top-k spatial-keyword queries [5, 14, 17, 25, 26]. These are classified into two approaches: 1) spatial-then-text and 2) text-then-spatial. The former (the latter) first builds the spatial (text) index, and then, builds the text (spatial) index for the objects in each group partitioned by the spatial (text) index. IR-tree [5, 14] is the representative for the former and S2I [17] for the latter. Figure 2a shows the structure of IR-tree. It groups objects by the location and creates an R-tree to manage the groups. It builds an inverted index on each node of the R-tree. Given a query, it traverses the R-tree referencing the inverted indices for the desired objects. Figure 2b shows the structure of S2I. It groups objects by the text and creates a keyword index to manage the groups. It

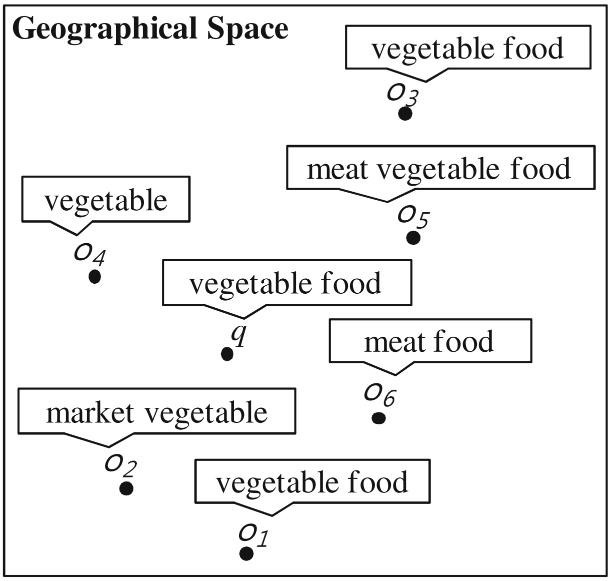
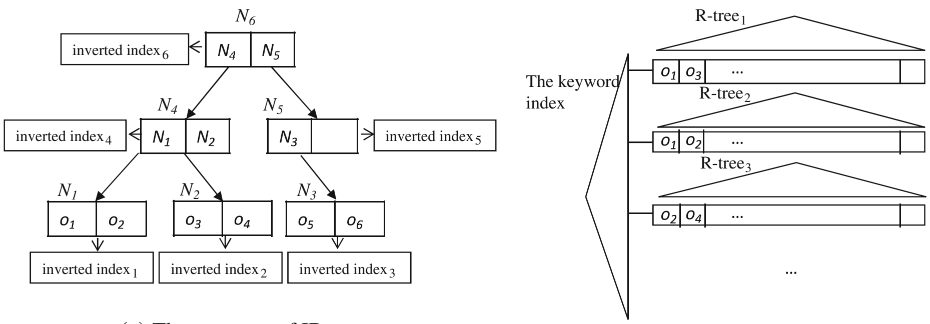


Figure 1 A sample data set for top-k spatial-keyword queries



(a) The structure of IR-tree.

(b) The structure of S2I.

Figure 2 Existing indexing methods

then builds an R-tree to manage objects associated with each keyword. Given a query, it traverses the inverted index referencing the R-trees for the desired objects.

1.1 Motivation

Existing methods overlooked one important issue: The performance of different index schemes often vary widely for different query loads. To see this, we characterize top-k spatial-keyword queries using three parameters: the value k, the value p ($0 \leq p \leq 1$), which captures users' preference for spatial proximity over textual relevancy, and the number of keywords in the query. First, we show that a particular index scheme may fit well only particular query loads.

QueryLoad 1 *For a larger p, IR-tree and S2I are more efficient.*
A larger p value indicates that the user has a stronger preference for spatial proximity. In IR-tree, the entire set of objects are clustered by the location in an R-tree while the inverted index is partitioned over multiple nodes of the R-tree. Consequently, for a larger p, IR-tree takes advantage of the clustering effect of the R-tree. In S2I, each keyword has an R-tree on the set of objects containing the keyword; for query processing, only R-trees for given query keywords are involved. As a result, as p becomes larger, S2I becomes more efficient due to the effect of clustering by spatial proximity of the R-trees.

QueryLoad 2 *When the number of query keywords is small, S2I is more efficient.*
S2I needs to access R-trees only for the given query keywords. Consequently, the number of indices involved decreases as the number of keywords in a query does.

Second, more importantly, there are query loads that none of the existing methods can handle efficiently. IR-tree and S2I are sub-optimal when k is very small or very large. We give two query loads in the following examples and show a simple separate index method outperforms existing methods.

QueryLoad 3 *When k is very small, IR-tree and S2I are sub-optimal.*
IR-tree and S2I partition the set of objects into groups by the location or text, i.e., leaf nodes or posting lists. To find the top-1 object, they process the data group by group.

Intuitively, however, since we are only interested in a small number of objects with the highest scores, we should just consider the objects that are ranked the top by both spatial proximity and textual relevancy. Thus, unlike IR-tree, which indexes location then text, or S2I, which indexes text then location, we may create indices for location or text separately. For each index, we obtain a sorted list according to spatial proximity or textual relevancy incrementally as much as we need for the given query. Then, for two sorted lists, we use Threshold Algorithm(TA) [6] *to find the top-k results. This way, we can stop the execution as soon as we get the top-k results while processing the data object by object without a need to process all the objects in a group as in IR-tree or S2I.*

QueryLoad 4 *When k is very large, IR-tree and S2I are sub-optimal.*

Some queries return a large number of objects. For example, an extreme case is retrieving all the objects in the order of the score. In this case, we need to access all the objects. However, since IR-tree and S2I access the objects in the order of the score, they require many random accesses due to index searching. A simple separate index method can achieve better performance. If we cluster objects according to their object IDs by each index, we access all the objects for each index in the order of the object ID by one sequential scan, and then, compute their combined scores by merging two lists.

1.2 Our contributions

We propose a generic model, which we call *G-Index Model*, for indexing top-*k* spatial-keyword queries. Our contributions are as follows: 1) G-Index Model is a unified framework. It exhaustively enumerates all the possible methods for top-*k* spatial-keyword queries. We conjecture that data clustering is the key element in composing various index schemes. Thus, we enumerate index schemes through combinations of different clustering techniques. We show that all the existing methods for top-*k* spatial-keyword queries map to those generated by G-Index Model. Specifically, the generated methods M3, M4, and M5 in Section 3 are mapped to RASIM [13], IR-tree [5, 14], and S2I [17], respectively. This shows that G-Index Model sheds new light on understanding the existing methods under a unified framework. Using G-Index Model, we also discover two new index schemes that have not been reported before. Those methods (M1 and M2 in Section 3) are the most efficient for extreme query loads such as QueryLoads 3 and 4 presented in Section 1.1. 2) We show that G-Index Model is generic. That is, G-Index Model can generate index schemes for a class of queries integrating arbitrary multiple types. For this, we show that G-Index Model can enumerate index schemes for two classes of queries: the spatial-keyword query (without the top-*k* constraint) and the top-*k* spatial-keyword-relational query, which adds the relational data type to the top-*k* spatial-keyword query. 3) We propose a cost model of the generated methods for the top-*k* spatial-keyword query and compare their performance. The proposed cost model can be used for efficient physical database design for top-*k* spatial-keyword queries by figuring out which method is the best for a certain usage pattern (i.e., a set of query loads and frequencies). We validate the cost model through extensive experiments by showing that the results are consistent with those from the cost model.

The rest of this paper is organized as follows. Section 2 explains preliminaries. Section 3 proposes G-Index Model. Section 4 shows the genericness of G-Index Model. Section 5 presents a cost model to estimate the query performance of the generated methods. Section 6 presents the experimental results. Section 7 describes the comparisons with related work. Section 8 summarizes and concludes the paper.

2 Preliminaries

2.1 Top-k spatial-keyword queries

Data and query definition Let D be a database. Each object o in D is a triple ($o.id$, $o.loc$, $o.doc$) where $o.id$ is the identifier of o, $o.loc$ is o's location in a multidimensional space, and $o.doc$ is a text description of o. A top-k spatial-keyword query q is defined as a quadruple ($q.loc$, $q.keywords$, $q.k$, $q.p$) where $q.loc$ is a location description, $q.keywords$ a set of keywords $\{k_1, k_2, ..., k_n\}$, $q.k$ the desired number of results, and $q.p$ the user preference of spatial proximity over textual relevancy.

Query semantics Top-k spatial-keyword queries retrieve k objects with the highest (or lowest) combined scores of spatial proximity and textual relevancy. Without loss of generality, in the rest of this paper, we assume that we are looking for objects having the highest scores. We adopt the following scoring function $S(q, o)$ [13].

$$\begin{aligned} S(q,o) &= q.p * S_{SP}(q.loc, o.loc)/maxSP \\ &\quad +(1-q.p) * S_{TR}(q.keywords, o.doc)/maxTR \end{aligned} \tag{1}$$

In (1), the *spatial component score* $S_{SP}(q.loc, o.loc)$ and the *textual component score* $S_{TR}(q.keywords, o.doc)$ are normalized by $maxSP$ and $maxTR$, respectively, where $maxSP$ ($maxTR$) is the largest possible spatial (keyword) score for objects in D.

The spatial component score $S_{SP}(q.loc, o.loc)$ computes the spatial proximity between an object and a query point. In this paper, we set $S_{SP}(q.loc, o.loc) = maxSP - ED(q.loc, o.loc)$, where ED is the Euclidean distance so that the higher the value of $S_{SP}(q.loc, o.loc)$ is, the more relevant o is to the query.

The textual component score $S_{TR}(q.keywords, o.doc)$ computes the textual relevancy between a set of query keywords and an object. In this paper, we adopt the *term-weighting scheme* [1], which is the most well known score for measuring textual relevancy:

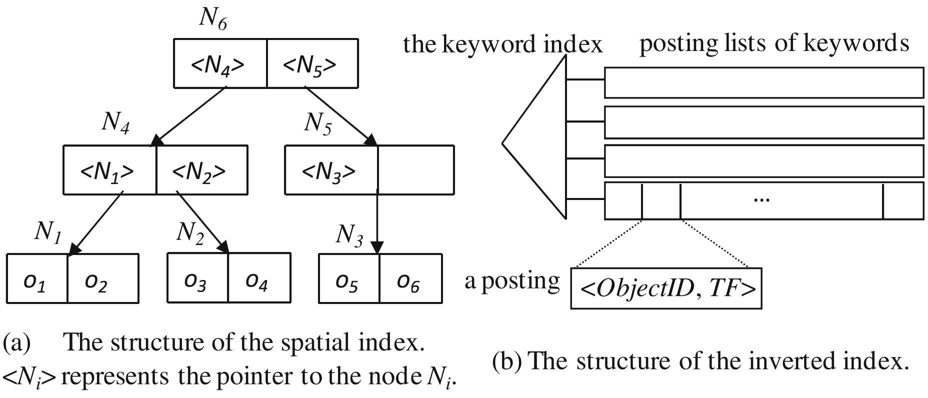
$$\begin{aligned} S_{TR}(q.keywords, o.doc) &= \sum_{i=1}^{n} S_{TR}(q.k_i, o.doc) \\ &= \sum_{i=1}^{n} TF(o.doc, k_i)\ log \frac{N}{DF(D, k_i)} \end{aligned} \tag{2}$$

Here, $TF(o.doc, k_i)$ is the term frequency of keyword k_i in $o.doc$; $DF(D, k_i)$ is the document frequency for keyword k_i in D; N is the number of objects in D. The higher the value of $S_{TR}(q.keywords, o.doc)$, the more relevant o is to the query.

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 both the spatial and the textual component scores.

2.2 Index structures

Efficient query processing of top-k spatial-keyword queries relies on smart use of the spatial index and the text index. The spatial index includes those in the R-tree family [2, 10, 19], those in the MBR-MLGF family [20, 21], and the quadtree [8]. They differ in the way they cluster the objects. Those in the R-tree family and the quadtree cluster objects in the original space while those in the MBR-MLGF family cluster objects in a transformed space. We can use any one of those indexes for the spatial index. Figure 3a shows a general



(a) The structure of the spatial index. $<N_i>$ represents the pointer to the node N_i. (b) The structure of the inverted index.

Figure 3 Index structures

structure of the spatial index, which consists of a multilevel directory and a set of leaf nodes. Each leaf node contains a set of objects clustered by the location. The spatial index groups the entire set of objects into the leaf nodes. Here, each leaf node in mapped to a disk page.

The inverted index is the most widely used index for text data [1]. Figure 3b shows its structure. It consists of a set of keywords, and each keyword is associated with a posting list. A posting in a posting list contains information such as the object identifier (*ObjectID*) and frequency of the keyword, i.e., *term frequency* (*TF*), in the document. In addition, a B+-tree (simply, the *keyword index*) can be built to search for the posting list of a specific keyword efficiently. Here, each posting list consists of one or more disk pages.

3 G-Index Model: a generic model of index schemes for top-*k* spatial-keyword queries

3.1 Overall concept

G-Index Model enumerates index schemes for top-*k* spatial-keyword queries from the viewpoint of clustering objects. Clustering is a key technique for efficient processing. *Clustering* means maintaining objects in the order of a certain criterion.[1] That is, clustering requires two components: 1) criteria for clustering (e.g., attributes of a relation) and 2) techniques for clustering (e.g., sorting). We call the former *clustering criteria*; the latter *clustering operators*. For example, if a relation $R(a, b)$ is stored as sorted on attribute a, R is "clustered" in attribute a [9]. Clustering allows us to scan the objects efficiently when the objects are accessed in the sequential way according to the order in which the objects are clustered. Here, we note that, in general, objects can be clustered in only a single criterion.

In top-*k* spatial-keyword queries, multiple attributes such as location or keyword are involved. Thus, we need to support clustering in multiple criteria. In G-Index Model, we support clustering in multiple criteria by employing the technique, *partitioning*, that groups objects according to a criterion. Partitioning allows us to cluster objects according to more

[1]In this paper, clustering is related to physical database design (i.e., allocation of contiguous storage in databases) not but to a data mining technique.

than one criterion, i.e., to cluster groups of objects by one criterion and to cluster the objects in each group by another criterion.

G-Index Model exhaustively enumerates all the possible index schemes by applying the possible clustering criteria and operators to each instance of clustering induced by partitioning.

Clustering criteria We define four clustering criteria for top-k spatial-keyword queries: (1) *location*, (2) *text*, (3) *object ID*, or (4) *score*. We can cluster objects by any feature/attribute of the objects, i.e., location, text, or object ID. Thus, objects with close location similarity, or objects with similar text, or objects with adjacent IDs will be clustered together independent of the query.[2] On the other hand, the scores are determined on the query. Thus, we need to cluster objects (i.e., results) based on the score dynamically since we can calculate their scores only after the query is given. In this case, we take advantage of clustering by the location (or text) since the objects with similar locations (or texts) also have similar scores. Specifically, since these objects were already clustered in the same group (e.g., a disk page) before the query is given, we need to cluster only the objects in the groups that are relevant to the given query based on the location and text by the score.

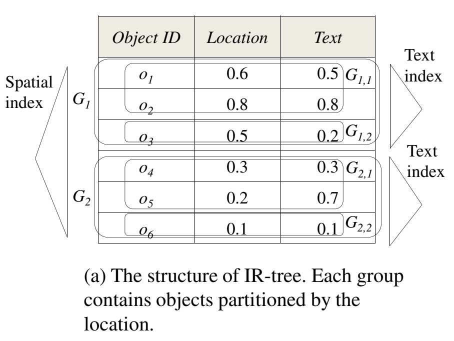
Clustering operators We define two clustering operators for top-k spatial-keyword queries: 1) $sorting(criterion)$ and 2) $indexing(criterion)$. The former sorts the input data by $criterion$; the latter creates an index for the input data on $criterion$. Both maintain a linear order of objects according to a criterion. We identify these operators from the observation of the existing methods. In Example 1, we show that existing methods can be modeled as combinations of clustering.

Example 1 Let us consider IR-tree and S2I to show how clustering is actually used. Figure 4a shows clustering used in the IR-tree.[3] First, it *partitions* the entire set of objects into groups (i.e., G_1 and G_2) by *location*, and it *indexes* the groups by *location*. Then, it *partitions* each group into sub-groups (i.e., $G_{1,1}$, $G_{1,2}$, $G_{2,1}$, and $G_{2,2}$) by *text*, and it *indexes* the sub-groups by *text*. Finally, the objects in each sub-group are *sorted* by *score* when the query is given. Figure 4b shows clustering used in S2I. First, it *partitions* the entire set of objects into groups (i.e., G_1 and G_2) by *text* where each group contains objects having a specific keyword, and it *indexes* the partitioned groups by *text*. Then, it *partitions* each group into sub-groups (i.e., $G_{1,1}$, $G_{2,1}$, $G_{2,2}$, and $G_{2,3}$) by *location*, and it *indexes* the sub-groups by *location*. Finally, the objects in each sub-group are *sorted* by *score* when the query is given.

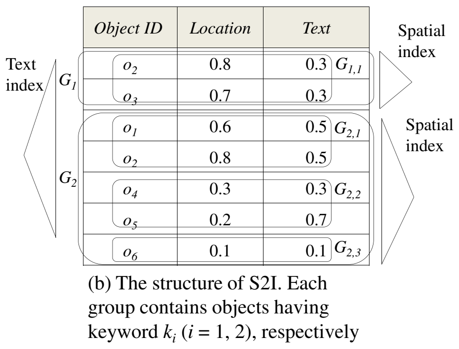
Exhaustive enumeration of index schemes Table 1 shows the index schemes enumerated by G-Index Model. Here, objects can be organized in two different ways: 1) *separate clustering* and 2) *combined clustering*. The former clusters the objects by either location or text separately; the latter clusters the objects by both location and text in a combined form. For simplicity, we denote the clustering only by the clustering criterion. For example, we denote *clustering*(*location*) simply by *location*. In $A \rightarrow B$, '$\rightarrow$' represents partitioning; 'A' or 'B' represents a clustering criterion, and either *indexing*() or *sorting*() can be applied to each

[2]To distinguish similarity between objects from relevancy between the object and the query, we call the former location similarity (textual similarity) and the latter spatial proximity (textual relevancy).

[3]For simplicity, we represent locations and texts of objects as floating numbers from 0 to 1.



(a) The structure of IR-tree. Each group contains objects partitioned by the location.



(b) The structure of S2I. Each group contains objects having keyword k_i ($i = 1, 2$), respectively

Figure 4 Clustering used in the existing index schemes

clustering criterion. Hence, '$A \rightarrow B$' means partitioning the data based on A, clustering the groups by A, and clustering the objects in each group by B. The salient point of G-Index Model is encompassing all the existing methods for top-k spatial-keyword queries. For example, methods (7) and (8) in Table 1 correspond to IR-tree and S2I, respectively. We describe details of the enumeration in Section 3.2 and show that all the existing methods map to the enumerated methods in Section 3.3

G-Index Model consists of two step processes: 1) index scheme enumeration and 2) index scheme refinement. In Section 3.2, we describe index scheme enumeration. In Section 3.3, we describe index scheme refinement that 1) prunes infeasible or definitely inefficient index schemes from those enumerated and that 2) sub-divides each of the remaining index schemes depending on whether each clustering instance is feasible in that index scheme.

3.2 Index scheme enumeration

Table 1 enumerates altogether six different clustering combinations (we call them methods) for separate clustering, and altogether six for combined clustering. In separate clustering, we need to merge the objects relevant to the query based on the location with those based on the text. Here, clustering the objects by object ID can facilitate efficient merging. Thus, separate clustering involves three clustering criteria: 1) location (or text), 2) object ID, and 3) score. For combined clustering, there is no need to cluster the objects by object ID,

Table 1 Enumerated index schemes by combining clustering

Separate Clustering	Combined Clustering
(1) $location \vert text \rightarrow ID \rightarrow score$	(7) $location \rightarrow text \rightarrow score$
(2) $location \vert text \rightarrow score \rightarrow ID$	(8) $text \rightarrow location \rightarrow score$
(3) $ID \rightarrow score \rightarrow location \vert text$	(9) $location \rightarrow score \rightarrow text$
(4) $ID \rightarrow location \vert text \rightarrow score$	(10) $text \rightarrow score \rightarrow location$
(5) $score \rightarrow location \vert text \rightarrow ID$	(11) $score \rightarrow location \rightarrow text$
(6) $score \rightarrow ID \rightarrow location \vert text$	(12) $score \rightarrow text \rightarrow location$

as merging is not needed. Thus, combined clustering involves three clustering criteria: 1) location, 2) text, and 3) score.

Separate clustering has two separate lists for the location and text. The method *clustering*(*location*|*text*) → *clustering*(*ID*) → *clustering*(*score*) first partitions the entire set of objects into groups and clusters the partitioned groups based on the location (or text). Then, it partitions each group into sub-groups and clusters sub-groups based on ID for merging two lists. Finally, it clusters objects in each sub-group based on the score. On the other hand, combined clustering does not need merging. Thus, the method *clustering*(*location*) → *clustering*(*text*) → *clustering*(*score*) first partitions the entire set of objects into groups and clusters the partitioned groups based on the location. Then, it partitions each group into sub-groups and clusters the sub-groups based on the text. Finally, it clusters the objects in each sub-group based on the score.

3.3 Index scheme refinement

3.3.1 Pruning step

The pruning step consists of two sub-steps: 1) pruning the whole methods that are definitely less efficient than the other methods in Table 1 and 2) pruning inefficient/infeasible clustering operators within each method.

The first sub-step is as follows. Many methods in Table 1 are inherently inefficient regardless of the query load. This is not difficult to understand: A query comes with spatial and textual constraints, and clustering based on the location and/or text limits the search space for the objects relevant to the query. Intuitively, we should cluster objects first by these criteria, and then, by other criteria. Clearly, (3) and (4) are not very meaningful. Similarly, (5), (6), and (9)–(12) are not meaningful either.

The second sub-step is as follows. By looking at the characteristics of clustering operators and criteria, we derive the following relationships for each clustering criterion: (1)$indexing$ (*ID*) is less efficient than $sorting$ (*ID*), (2) $sorting$ (*location*) is less efficient than $indexing$ (*location*), (3) $sorting$ (*text*) is less efficient than $indexing$ (*text*), and (4) $indexing$ ($score$) is infeasible. Relationship (1) stems from the fact that, since the object ID is used for merging, we need to access all the objects retrieved from each index in the order of the object ID. Consequently, a sequential scan over the objects sorted by the object ID will be more efficient than a scan through an index. Relationships (2) and (3) stem from the fact that the location and text are used for retrieving selectively only those objects relevant to the given query; thus, in comparison, sorting the entire set of objects is unnecessary and inefficient. This is further justification since spatial or textual constraints are usually very selective. Relationship (4) stems from the fact that we cannot create an index based on the final score (which is a derived attribute).

By applying the best operator for each criterion to the enumerated methods (1) and (2) and (7) and (8) in Table 1, we obtain the methods in Table 2.

Table 2 Generated methods

Method1. $indexing$ (*location*|*text*) → $sorting$ (*ID*) → $sorting$ (*score*)
Method2. $indexing$ (*location*|*text*) → $sorting$ (*score*) → $sorting$ (*ID*)
Method3. $indexing$ (*location*) → $indexing$ (*text*) → $sorting$ (*score*)
Method4. $indexing$ (*text*) → $indexing$ (*location*) → $sorting$ (*score*)

3.3.2 Sub-dividing step

We now sub-divide each method in Table 2 to two methods according to whether each clustering in the method is indeed used or not. Specifically, $indexing$ (*location*) and $indexing$ (*text*) always produce the results in the unit of groups since the index is built on the partitioned groups (i.e., leaf nodes in the spatial index and posting lists in the inverted index). In this case, we can apply subsequent clustering to the set of objects or sub-groups in each group. In contrast, $sorting$ (*ID*) and $sorting$ (*score*) can optionally choose the unit to produce the results: 1) in the unit of groups (partitioning is done in the unit of groups) or 2) in the unit of objects (partitioning in done in the unit of objects). For the former, we can apply subsequent clustering to the set of objects in each sub-group; for the latter, we cannot further apply it since the results are already in the unit of objects. Consequently, the subsequent clustering cannot be actually used.

We sub-divide Method1 in Table 2 into two methods according to the unit that $sorting$(*ID*) produces. Thus, we can generate the following methods: (a) $indexing$(*location*|*text*) → $sorting$(*ID*) and (b) $indexing$(*location*|*text*) → $sorting$ (*ID*) → $sorting$(*score*). However, in (b), since the criterion *ID* does not contribute to limiting the search space, all the groups partitioned by *ID* still need to be processed, obviating the need for subsequent clustering in each sub-group partitioned by ID. Consequently, (b) is reduced to (a). As a result, Method1 in Table 2 generates a method, called *separate clustering with merging* (simply, *M1*).

M1 $indexing($location|text$) \rightarrow sorting\,(ID)$

Figure 5a shows M1. M1 first clusters the objects based on the location or text separately using $indexing$(). Then, for each index, it clusters the objects in each group based on the object ID using $sorting$().

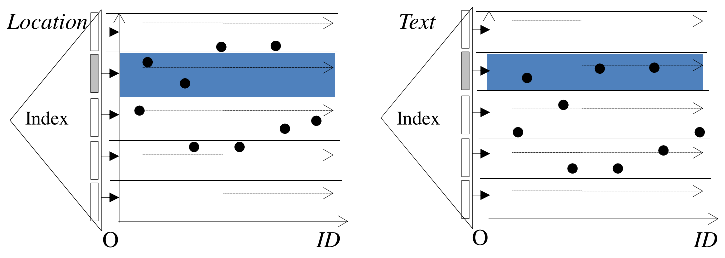
We sub-divide Method2 in Table 2 into two methods according to the unit that $sorting$(*score*) produces. Thus, we generate the following methods: (a) $indexing$ (*location*|*text*) → $sorting$ (*score*) and (b) $indexing$ (*location*|*text*) → $sorting$ (*score*) → $sorting$ (*ID*). The former provides top-k pruning in the unit of objects; the latter provides top-k pruning in the unit of groups. As a result, Method2 in Table 2 generates a method called *separate clustering with object merging and ranking* (simply, *M2*) and another method, called *separate clustering with group merging and ranking* (simply, *M3*), which is mapped to RASIM [13].

M2 $indexing($location|text$) \rightarrow sorting\,(score)$

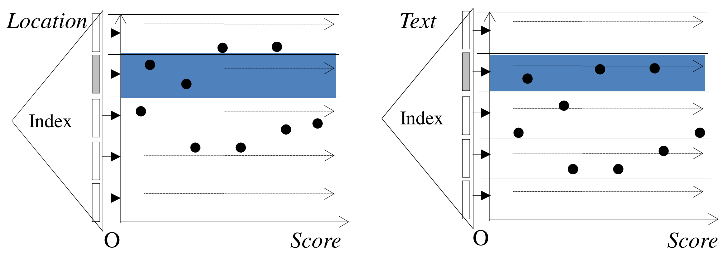
M3 $indexing($location|text$) \rightarrow sorting(score) \rightarrow sorting(ID)$

Figure 5b shows M2; Figure 5c M3. M2 first clusters the objects based on the location or text separately using $indexing$(). Then, for each index, it clusters the objects in each group based on the score using $sorting$(). M3 first clusters the objects based on the location or text separately using $indexing$(). Then, for each index, it clusters the sub-groups in each group based on the score using $sorting$(). Last, it clusters the objects in each sub-group based on the object ID using $sorting$().

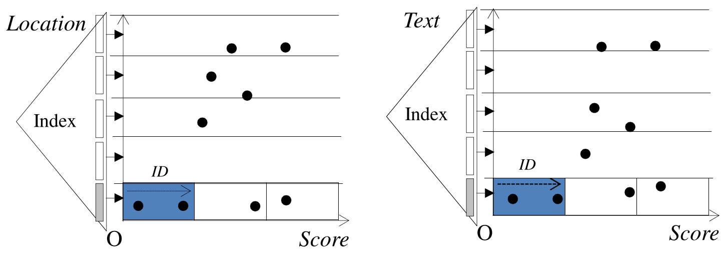
Method 3 in Table 2 directly generates a method, called *combined clustering with space first* (simply, *M4*), which is mapped to IR-tree [5, 14], and Method 4 in Table 2 directly generates a method, called *combined clustering with keyword first* (simply, *M5*),



(a) Separate clustering with merging (M1).



(b) Separate clustering with object merging and ranking (M2).



(c) Separate clustering with group merging and ranking (M3).

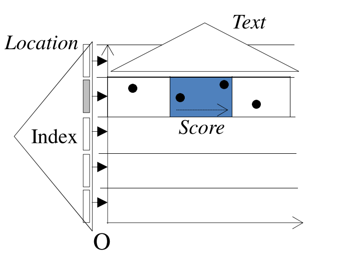
Figure 5 Separate clustering approach

which is mapped to S2I [17]. For each rule, since $indexing(location)$ and $indexing(text)$ always produce the results in the unit of groups, we can apply their subsequent clustering $sorting(score)$ to the results of $indexing(location)$ or $indexing(text)$. Thus, there is no further sub-division.

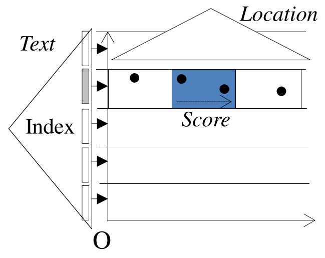
M4 $indexing(\text{location}) \rightarrow indexing(\text{text}) \rightarrow sorting(score)$

M5 $indexing(\text{text}) \rightarrow indexing(\text{location}) \rightarrow sorting(score)$

Figure 6a shows M4; Figure 6b M5. M4 first clusters the objects based on the location using $indexing()$. Then, it clusters the sub-partitioned groups in each group based on the text using $indexing()$. Last, it clusters the objects in each sub-group based on the score



(a) Combined clustering with space first (M4).



(b) Combined clustering with keyword first (M5).

Figure 6 Combined clustering approach

using $sorting()$. M5 apply the clustering operators similarly where the roles of the location and text are exchanged.

3.4 Implementation

M1 M1 *indexes* the entire set of objects according to the location or text separately through the spatial index and the inverted index. From each index, it retrieves all the objects that can be the results based on spatial proximity or textual relevancy *as sorted by the object ID*. We define those objects as *the result candidates* (simply, the set O_{RC}). M1 stores the objects in each leaf node of the spatial index as sorted by the object ID; so are $|O_{RC}|$ top objects (according to textual relevancy) of the posting list for each keyword. When the queries are given, we construct O_{RC} for the spatial index by accessing the leaf nodes within the relevant region[4] to retrieve top $|O_{RC}|$ objects for the query point and by merging the objects in the nodes based on the object ID; we construct O_{RC} for the inverted index by merging the postings for the query keywords based on the object ID.

Query processing steps for M1 are as follows. 1) We retrieve O_{RC} from the spatial index (inverted index) and stores them in $RC\_SPATIAL$ (RC_INV). 2) We merge $RC\_SPATIAL$ and RC_INV based on the object ID and store them into $MERGED$. 3) We calculate the combined scores of the objects in $MERGED$ and insert them into a priority queue PQ, which maintains the objects in the order of the combined score.[5] 4) We retrieve top-k results from PQ in the order of the combined score.

The important part of M1 is how to set $|O_{RC}|$. We need to preset $|O_{RC}|$ regardless of queries. However, it is difficult to set $|O_{RC}|$ to guarantee the correctness of the results since $|O_{RC}|$ will be different depending on the data distribution and query loads. We set $|O_{RC}|$ under the assumption of uniform distribution of data, allowing possible incorrect results for other distributions. With the uniform distribution of the sorted lists, Pang et al. [16] have

[4]We cannot know an exact region that retrieves $|O_{RC}|$ objects for every query point. Thus, as the region, we use a circle where its area is equal to $\frac{EntireSpace \cdot |O_{RC}|}{N}$ and enlarge the circle incrementally until $|O_{RC}|$ objects are retrieved.

[5]The sorting used in this step needs to be differentiated from *sorting*() defined as a clustering operator. The *sorting*() operator is associated with the previous partitioning. That is, *sorting*() is applied to each sub-group resulting by the partitioning. However, this step simply sorts the entire set of objects that are merged from all the groups.

claimed that $(m \times q.k^{1/m} \times N^{(m-1/m)})$ object accesses from each list, where m is the number of lists, guarantee finding exact top-k results. We obtain $|O_{RC}|$ by substituting the parameters for each dataset (i.e., m is 2, $q.k$ is 1000, which is the maximum number of $q.k$ used in most search engines, and N is the size of the dataset) for this equation.[6]

M2 M2 *indexes* the entire set of objects according to the location or text separately through the spatial index and the inverted index. From each index, it retrieves the list of the objects *as sorted by the score*. For the spatial component, M2 retrieves the objects in the sorted order from the spatial index by applying Incremental NN [12]. For the keyword component, it can retrieve the objects in the sorted order by a simple scan over the posting list of each keyword from the inverted index. This is possible since the objects in each posting list have been stored in the order of the score, which can be determined independent of the query. When the query keywords are given, M2 obtains the list of the objects sorted by the score by applying Threshold Algorithm(TA) [6] to the multiple sorted lists for the keywords. Thus, we have two sorted lists—one for the spatial component and the other for the keyword component. To process a query in M2, we apply TA to the sorted lists. Whenever we retrieve an object from each list, we update the threshold value of TA as the score obtained by combining the upper bound of the component scores of the objects that have not yet been retrieved from each list.

M3 M3 *indexes* the objects and processes the query in a way similar to that of M2. The only difference is that M3 retrieves the objects in the unit of groups *as sorted by the (group) score*, and each group contains the objects *as sorted (and stored) by the object ID*. We use the group as the unit of physical access mapped to one disk page. Hence, we use the objects in a leaf node of the spatial index as a group and the objects in a disk page of the posting list as a group for the sake of fairness with the spatial index. To process a query, M3 retrieves the sorted list of the groups (i.e., leaf nodes) from the spatial index according to the score by extending Incremental NN [12] *in the unit of the leaf node* instead of the individual object. Here, we use the maximum score of the objects in the node as the score of the node. For the keyword component, M3 retrieves the groups in the posting list of each keyword as sorted by the score where the objects have been stored in each group as sorted by the object ID. When the query keywords are given, it obtains the list of the groups sorted by the score by extending TA [6] *in the unit of the group* instead of the individual object.

M4 The index structure for M4 builds the spatial index on the entire set of objects and builds an inverted index on each leaf node of the spatial index. Each leaf node has an inverted index for the objects in the leaf node; each internal node has an inverted index on their child nodes. Query processing in M4 is done by extending Incremental NN [12] to use the combined score rather than spatial proximity only.

M5 The index structure for M5 builds the inverted index on the entire set of objects and builds a spatial index on each posting list of the inverted index. Each entry of a leaf node of the spatial index also includes TF, which represents the text, for the object; that of an internal node of the spatial index includes the maximum TF of the objects in the child nodes.

[6]In all our experiments including real data sets, all the results were correct.

Query processing for M5 is done by extending Incremental NN [12] to the multiple spatial indexes for the given keywords rather than one spatial index.

4 Genericness of G-Index Model

We show that G-Index Model is generic, i.e., it can be applied to a class of queries integrating arbitrary multiple data types. First, we show that G-Index Model can enumerate index schemes for spatial-keyword queries (without the top-k constraint) by removing the 'score' from the set of the clustering criteria for top-k spatial-keyword queries since the criterion 'score' is needed for supporting top-k pruning. As a result, we obtain the following four methods: (a) location|text → ID (simply, *SK-M1*), (b) ID → location|text (simply, *SK-M2*), (c) location → text (simply, *SK-M3*), and (d) text → location (simply, *SK-M4*). We will show that all the existing methods for spatial-keyword queries map to these methods in Section 7.

Second, we show that G-Index Model can enumerate index schemes for new classes of queries involving new data types. Let us consider a new class of queries that adds the relational data type to the top-k spatial-keyword query. Let us call it the *top-k spatial-keyword-relational query*. G-Index Model enumerates index schemes for this query by adding the relational data type as a new clustering criterion. 1) We enumerate the index schemes through the combination of clustering according to the index scheme enumeration in Section 3.2. Here, we consider five clustering criteria: location, text, relational, object ID, and score. 2) We prune index schemes that are not feasible according to the pruning step of the index scheme refinement in Section 3.3.1. As a result, we obtain the index schemes for the top-k spatial-keyword-relational query as in Table 3. For example, in Method (1), we find the objects based on each clustering criteria separately, i.e., location, text, and relational. Then, we merge them based on the object ID. Finally, we find the results according to the score. We leave detailed methods and their analysis as a further study.

5 Cost analysis

We present a cost model with regard to storage overhead in Section 5.2 and query processing cost in Section 5.3 for the generated index schemes. Since the index is constructed before the queries are given, we only analyze the query processing cost, but do not consider the index construction cost. In Section 5.4, using the cost model, we show how to do physical database design by selecting the best method for a usage pattern. We verify the results through experiments in Section 6.

Table 3 Generated index schemes for the top-k spatial-keyword-relational query

(1) *location*|*text*|*relational* → *ID* → *score*	Separate clustering
(2) *location*|*text*|*relational* → *score* → *ID*	
(3) *location* → *text* → *relational* → *score*	Combined clustering
(4) *location* → *relational* → *text* → *score*	
(5) *text* → *location* → *relational* → *score*	
(6) *text* → *relational* → *location* → *score*	
(7) *location* → *text* → *relational* → *score*	
(8) *location* → *relational* → *text* → *score*	

5.1 Assumptions and notations

We define a query load for top-k spatial-keyword queries by controlling three query parameters: 1) $q.k$, 2) $q.p$, and 3) $|q.keywords|$. We devise the cost model for query processing in Section 5.3 under the assumption that it is proportional to the number of pages accessed while taking into account the effect of sequential/random accesses to disk pages. In effect, the cost model represents the equivalent number of random accesses to the disk pages (i.e., we transform the number of sequential accesses to an equivalent number of random accesses by considering the ratio[7] of the cost of a sequential access to that of a random access). Thus, we can roughly estimate the query processing time by multiplying the cost of a random access to the cost obtained in Section 5.3. For simplicity, we assume that data has a uniform distribution. We summarize the notation used for the cost model in Table 4.

5.2 Analysis of storage overhead

The separate clustering methods, i.e., M1, M2, M3, have the same storage overhead. Their storage overhead, S_{M123}, consists of those of 1) the spatial index, $S_{M123,SPATIAL}$, and 2) the inverted index, $S_{M123,INV}$. The height of the spatial index is $\lceil \log_{BF} N \rceil$, and each level i of the index has BF^i nodes.[8] $S_{M123,INV}$ consists of the costs of 1) the posting lists, $S_{M123,POS}$, and 2) the keyword index, $S_{M123,KEY}$. The number of disk pages needed to store the posting lists is $\lceil |W_T|/BF \rceil$. The height of the keyword index of the inverted index is $\lceil \log_{BF} |W| \rceil$, and each level i of the index has BF^i nodes. Thus, S_{M123} is estimated as in (3).

$$\begin{aligned} S_{M123} &= S_{M123,SPATIAL} + S_{M123,POS} + S_{M123,KEY} \\ &= \sum_{i=0}^{\lceil \log_{BF} N \rceil -1} BF^i + \left\lceil \frac{|W| \cdot P_{AVG}}{BF} \right\rceil + \sum_{i=0}^{\lceil \log_{BF} |W| \rceil -1} BF^i \\ &\approx \frac{N}{BF} + \frac{|W_T|}{BF} + \frac{|W|}{BF} \end{aligned} \tag{3}$$

The storage overhead for M4, S_{M4}, consists of those of 1) the spatial index, $S_{M4,SPATIAL}$, and 2) the inverted index, $S_{M4,INV}$. $S_{M4,SPATIAL}$ is the same as $S_{M123,SPATIAL}$. M4 creates an inverted index for each node of the spatial index, and hence, its number is the same as that of the nodes in the spatial index. Here, we estimate the size of each inverted index as that of the original inverted index (i.e., $S_{M123,INV}$) divided by the number of leaf nodes (i.e., $\lceil N/BF \rceil$) since $S_{M123,INV}$ is partitioned to leaf nodes of the spatial index. We divide $S_{M4,INV}$ into the inverted indices for leaf nodes and those for internal nodes. The total space of the former is the same as $S_{M123,INV}$. We note that the space for the latter (i.e., the inverted indices for internal nodes = $\sum_{i=0}^{\lceil \log_{BF} N \rceil -2} BF^i \cdot \frac{S_{M123,INV}}{\lceil N/BF \rceil}$) must be added to S_{M4}.

[7] We set the ratio to be $\frac{C_S}{C_R}$ where $C_S (C_R)$ is the cost for a sequential (random) access to a disk page.

[8] BF will be rather different for each type of the index. However, for simplicity, we assume that BFs for all indices are the same.

Table 4 The Notation

Symbols	Definitions
N	the total number of objects
$\|W\|$	the number of unique keywords in the collection
$\|W_T\|$	the total number of keywords in the collection
$\|P_{Wi}\|$	the number of postings for keyword W_i
$\|P_{AVG}\|$	the average number of postings per keyword (i.e., $\|W_T\|/\|W\|$)
$\|O_{RC}\|$	the number of result candidates
$\|O_{Prune}\|$	the number of the objects to access until top-k pruning
BF	blocking factor (i.e., the maximum number of entries stored in a disk page) of the index pages
$C_S(C_R)$	the cost for a sequential (random) access to a disk page

Thus, we obtain

$$\begin{aligned} S_{M4} &= S_{M4,SPATIAL} + S_{M4,INV} \\ &\approx S_{M4,SPATIAL} + (S_{M123,INV} + \left(\sum_{i=0}^{\lceil \log_{BF} N \rceil - 2} BF^i \cdot \left(\frac{S_{M123,INV}}{\lceil N/BF \rceil} \right) \right) \\ &= S_{M123} + \sum_{i=0}^{\lceil \log_{BF} N \rceil - 2} BF^i \cdot \frac{S_{M123,INV}}{\lceil N/BF \rceil} \\ &\approx S_{M123} + \frac{N}{BF^2} \cdot \frac{|W_T| + |W|}{N} \end{aligned} \tag{4}$$

The storage overhead for M5, S_{M5}, consists of those of 1) the spatial index, $S_{M5,SPATIAL,}$ and 2) the inverted index, $S_{M5,INV}$. M5 creates a spatial index for each keyword W_i. $S_{M5,SPATIAL}$ is calculated by summing up the space for the spatial indexes for all the keywords. $S_{M5,INV}$ is equal to $S_{M123,INV}$. Thus, S_{M5} is estimated as in (5). We note that $S_{M5,SPATIAL}$ depends on $|W_T|$, not on N. This means that the same object is indexed by multiple spatial indexes for the keywords contained in the object. Thus, S_{M5} is always much bigger than S_{M123} since so is $|W_T|$ than N.

$$\begin{aligned} S_{M5} &= S_{M5,SPATIAL} + S_{M5,INV} \\ &= \sum_{i=1}^{|W|} \sum_{j=0}^{\lceil \log_{BF} |P_{Wi}| \rceil - 1} BF^j + S_{M123,INV} \\ &\approx |W| \cdot \sum_{j=0}^{\lceil \log_{BF} |P_{AVG}| \rceil - 1} BF^j + \frac{|W_T| + |W|}{BF} \\ &\approx \frac{|W| \cdot |P_{AVG}|}{BF} + \frac{|W_T| + |W|}{BF} \end{aligned} \tag{5}$$

5.3 Analysis of query processing cost

The query processing cost for M1, $Proc_{M1}$, consists of 1) the cost for accessing the spatial index, $Proc_{M1,SPATIAL,}$ and 2) that for accessing the inverted index, $Proc_{M1,INV}$. M1 accesses $|O_{RC}|$ objects for each index. Approximately, a fraction $\frac{|O_{RC}|}{N}$ of $S_{M123,SPATIAL}$

is accessed; a fraction $\frac{|q.keywords|\cdot|O_{RC}|}{W_T}$ of $S_{M123,POS}$ and a fraction $\frac{|q.keywords|}{W}$ of $S_{M123,KEY}$ are accessed. Since M1 accesses $|O_{RC}|$ objects for each index at a time, it can take advantage of the sequential access to the inverted index in contrast to the other methods; we reflect its effect to $Proc_{M1,INV}$ by multiplying $\frac{C_S}{C_R}$.

$Proc_{M1}$ is estimated as in (6).

$$\begin{aligned} Proc_{M1} &= Proc_{M1,SPATIAL} + Proc_{M1,INV} \\ &\approx \frac{|O_{RC}|}{N} \cdot S_{M123,SPATIAL} + \frac{C_S}{C_R} \cdot \frac{|q.keywords| \cdot |O_{RC}|}{|W_T|} \cdot S_{M123,POS} \\ &\quad + \frac{|q.keywords|}{|W|} \cdot S_{M123,KEY} \\ &= \frac{|O_{RC}|}{BF} \cdot \left(1 + \frac{C_S}{C_R} \cdot |q.keywords|\right) + \frac{|q.keywords|}{BF} \end{aligned} \tag{6}$$

The query processing cost for M2, $Proc_{M2}$, consists of 1) the cost for accessing the spatial index, $Proc_{M2,SPATIAL}$, and 2) that for accessing the inverted index, $Proc_{M2,INV}$. M2 accesses $|O_{Prune}|$ objects for each index. Approximately, a fraction $\frac{|O_{Prune}|}{N}$ of $S_{M123,SPATIAL}$ is accessed; a fraction $\frac{|q.keywords|\cdot|O_{Prune}|}{W_T}$ of $S_{M123,POS}$ and a fraction $\frac{|q.keywords|}{W}$ of $S_{M123,KEY}$ are accessed. The maximum value for $|O_{Prune}|$ is calculated by $m \times N^{(m-1)/m} \times q.k^{1/m}$ where m is the number of sorted lists [16]. Here, $m = 2$ (one for spatial proximity and one for textual relevancy). Meanwhile, M2 provides top-k pruning in the unit of the object while the other methods in the unit of the group; we reflect this by multiplying $f_{M2} = \frac{|O_{Prune}|/BF}{\lceil |O_{Prune}|/BF \rceil}$. $Proc_{M2}$ is estimated as in (7).

$$\begin{aligned} Proc_{M2} &= Proc_{M2,SPATIAL} + Proc_{M2,INV} \\ &\approx f_{M2} \cdot \left(\frac{|O_{Prune}|}{N} \cdot S_{M123,SPATIAL} + \frac{|q.keywords| \cdot |O_{Prune}|}{|W_T|} \cdot S_{M123,POS}\right) \\ &\quad + \frac{|q.keywords|}{|W|} \cdot S_{M123,KEY} \\ &= f_{M2} \cdot \left(\frac{|O_{Prune}|}{BF} \cdot (1 + |q.keywords|)\right) + \frac{|q.keywords|}{BF} \end{aligned} \tag{7}$$

The query processing cost for M3, $Proc_{M3}$, consists of 1) the cost for accessing the spatial index, $Proc_{M3,SPATIAL}$, and 2) that for accessing the inverted index, $Proc_{M3,INV}$. The only difference of M3 from M2 is that M3 processes all the objects in each disk page at a time, obviating the need for f_{M2} in $Proc_{M2}$. $Proc_{M3}$ is estimated as in (8).

$$\begin{aligned} Proc_{M3} &= Proc_{M3,SPATIAL} + Proc_{M3,INV} \\ &\approx \frac{|O_{Prune}|}{BF} \cdot (1 + |q.keywords|) + \frac{|q.keywords|}{BF} \end{aligned} \tag{8}$$

The query processing cost for M4, $Proc_{M4}$, consists of 1) the cost for accessing the spatial index, $Proc_{M4,SPATIAL}$, and 2) that for accessing the inverted index, $Proc_{M4,INV}$. Compared with M3, M4 requires additional access overhead for inverted indices on internal nodes (i.e., $\frac{|W|+|W_T|}{BF^2}$). Approximately, a fraction $\frac{|O_{Prune}|}{N}$ of inverted indices on internal nodes is accessed. We use the same $|O_{Prune}|$ in M2 and M3 for M4 and M5 as well due to the following reason. M4 (M5) has a list sorted by spatial proximity (textual relevancy) and another list partitioned by spatial proximity (textual relevancy) where each group is sorted by spatial proximity (textual relevancy). Hence, we can conceptually consider M4 and M5 have two sorted lists. Meanwhile, we note that $Proc_{M4}$ is affected by $q.p$. This stems from the fact that M4 clusters the entire set of objects by a spatial index while clustering the partitioned groups by multiple text indices. Hence, if a high weight for spatial proximity is given (i.e., a high value of $q.p$), $Proc_{M4}$ decreases due to the clustering effect of the spatial index. We reflect the effect approximately by multiplying $f(q.p)$. Here, we use $(\frac{1-q.p}{0.5})$ for $f(q.p)$ so that $Proc_{M4}$ decreases for a high value of $q.p$ and increases for a low value of $q.p$ balancing at $q.p = 0.5$. $Proc_{M4}$ is estimated as in (9). We note that the additional overhead of M4 over M3 is supported by the fact that the performance of M3 is generally better than that of M4 except for the case of high $q.p$ in Section 6.

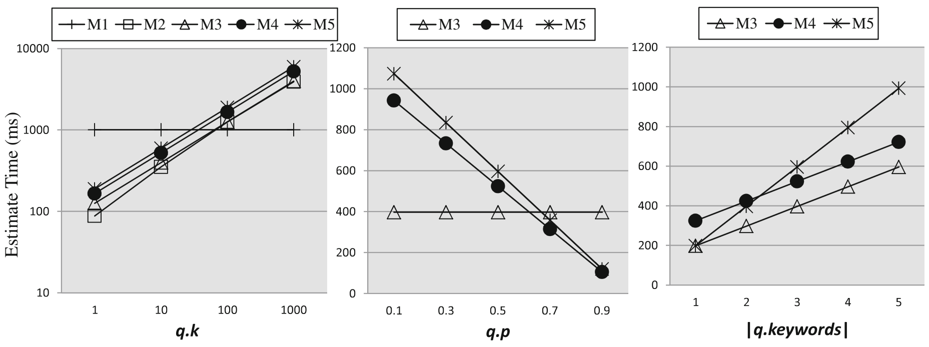
$$\begin{aligned} Proc_{M4} &= Proc_{M4,SPATIAL} + Proc_{M4,INV} \\ &= f(q.p) \cdot \left(Proc_{M3} + \frac{|O_{Prune}|}{N} \cdot \frac{|W| + |W_T|}{BF^2} \right) \\ &= f(q.p) \cdot \left(\frac{|O_{Prune}|}{BF} \cdot \left(1 + |q.keywords| + \frac{|W| + |W_T|}{N \cdot BF} \right) \right. \\ &\qquad \left. + \frac{|q.keywords|}{BF} \right) \end{aligned} \tag{9}$$

The query processing cost for M5, $Proc_{M5}$, consists of 1) the cost for accessing the spatial index, $Proc_{M5,SPATIAL}$, and 2) that for accessing the inverted index, $Proc_{M5,INV}$. Approximately, a fraction $\frac{|q.keywords| \cdot |O_{Prune}|}{W_T}$ of all the indices is accessed. For each keyword, M5 clusters the entire set of objects containing the keyword by a spatial index. Hence, for each query keyword, M5 is efficient for a high value of $q.p$ due to the effect of clustering by the spatial index as M4. $Proc_{M5}$ is estimated as in (10). We note that the performance of M5 degrades much as the number of query keywords increases since the steps must be repeated for each query keyword.

$$\begin{aligned} Proc_{M5} &= Proc_{M5,SPATIAL} + Proc_{M5,INV} \\ &\approx f(q.p) \cdot \left(\frac{|q.keywords| \cdot |O_{Prune}|}{|W_T|} \cdot (S_{M5,SPATIAL} + S_{M5,POS}) \right. \\ &\qquad \left. + \frac{|q.keywords|}{|W|} \cdot S_{M5,KEY} \right) \\ &= f(q.p) \cdot (2 \cdot |q.keywords| \cdot \frac{|O_{Prune}|}{BF} + \frac{|q.keywords|}{BF}) \end{aligned} \tag{10}$$

5.4 Physical database design

In this section, we show that our cost model allows us to do physical database design, i.e., to find the optimal method for a given usage pattern. Figure 7 shows the query processing time estimated using the cost model presented in Section 5.3. Here, we use *DataSet1* (N = 78,260, $|W|$ = 206,636, and $|W_T|$ = 18,397,075) as the default data set. The other constants are as follows: BF = 200 (i.e., page size = 4KB and object size = 20B) and $\frac{C_S}{C_R}$ = 1/250 (C_R = seek time + 1/2 * rotation time + block transfer time, C_S = block transfer time; specifications of the disk used: seek time = 8.9ms, rotation time = 5.56ms, block transfer time = 0.045ms). Default query parameters are as follows: $q.k$ = 10, $q.p$ = 0.5, and $|q.keywords|$ = 3. Figure 7a shows the estimated query processing time as $q.k$ is varied. For a small $q.k$, M2 shows the best performance due to the processing unit (i.e., an object) smaller than those of the other methods as indicated by (7); for a large $q.k$, M1 is the best due to the effect of sequential access (i.e., C_S/C_R) as indicated by (6).



(a) The estimated time as *q.k* is varied (*q.p* = 0.5 and |*q.keywords*| =3).
(b) The estimated time as *q.p* is varied (*q.k* = 10 and |*q.keywords*| = 3).
(c) The estimated time as |*q.keywords*| is varied (*q.k* = 10 and *q.p* = 0.5).

Figure 7 The estimated query processing time using the cost model

Figure 7b shows the estimated query processing time as $q.p$ is varied. Since the separate clustering methods (i.e., M1, M2, and M3) show similar trends for the other parameters except for $q.k$, we present only M3 as the representative of the separate clustering methods in the analysis and experiments. For a small $q.p$, M3 is the best; for a large $q.p$, M4 and M5 are the best. The greater $q.p$ is, the better the performance of M4 and M5 are than those of the other methods as indicated by (9) and (10).

Figure 7c shows the estimated query processing time as $|q.keywords|$ is varied. $|q.keywords|$ affects the performance of M5 much more than those of the other methods as indicated by (10). For a small $|q.keywords|$, the performance of M5 is the best; for a large $|q.keywords|$, M3 is the best.

Example 2 Figure 8 shows two usage patterns consisting of query loads and their frequencies. Here, we use the same data set and constrains as in Figure 7. By substituting query parameters in the cost formulas obtained in Section 5.3, we obtain the following expected query processing time for each method for the usage pattern A: M1 = 1007.05ms, M2 = 256.66ms, M3 = 306.14ms, M4 = 502.54ms, and M5 = 606.27ms. Thus, the optimal method for the given usage pattern in Figure 8 is M2. We also obtain the following expected query processing time for each method for the usage pattern B: M1 = 1009.86ms, M2 = 699.66ms, M3 = 768.15ms, M4 = 287.41ms, and M5 = 386.22ms. Thus, in this case, the optimal method is M4.

6 Performance evaluation

6.1 Experimental data and environment

In this section, we have implemented all the methods and indices for an extensive experiment. We compare the index size and query performance of the generated methods. The index size of the separate clustering methods is the sum of the size of the spatial index and that of the inverted index while the index size of the combined clustering methods is that of

	Query Loads			Frequencies (%)
	q.k	*q.p*	|*q.keywords*|	
Query Load1	3	0.3	4	70
Query Load2	10	0.4	2	20
Query Load3	20	0.5	3	10

(a) Usage pattern *A*.

	Query Loads			Frequencies (%)
	q.k	*q.p*	|*q.keywords*|	
Query Load1	20	0.9	4	80
Query Load2	30	0.7	5	20

(b) Usage pattern *B*.

Figure 8 The example of usage patterns

the their index structures themselves. 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. *DataSet2* is created combining texts from the 20Newsgroups dataset[9] and locations from LA streets[10]. *DataSet2* is the same as that used by Cong et al. [5] and Rocha-Junior et al. [17]. In *DataSet3*, the spatial data are generated randomly, and the text data are from Web pages collected through Web crawling. We generated four data sets of varying sizes for *DataSet3*: 1K, 10K, 100K, and 1M. The data sets were generated by randomly selecting a Web page for each spatial object. We use *DataSet3* to measure the performance as the data size is varied and *DataSet1* and *DataSet2* for the other experiments. Table 5 shows the characteristics of these data sets.

We generate five query sets, where $|q.keywords|$ is 1, 2, 3, 4, and 5, respectively. Each query set consists of 100 queries for each data set. Query locations are randomly generated in the space of the data set. Query keywords are randomly selected from a set of keywords each of whose document frequencies is greater than one percent of total number of objects—excluding infrequent or unrealistic keywords. Table 6 shows the query parameters and values used in the experiments. The default values are shown in bold.

We conduct all the experiments on a Pentium4 3.6GHz Linux PC with 2.5GB of main memory. 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

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

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

Table 5 Characteristics of the data sets

Data sets	*DataSet1*	*DataSet2*	*DataSet3*
Total number of objects	78,260	131,461	1,000,000
Maximum number of postings per keyword	55,949	131,461	794,915
Average number of postings per keyword	89	168	818
Total number of unique words in the data set	206,636	114,831	318,916
Total number of words in the data set	18,397,075	19,278,878	261,040,074

for storing data and indices. The page size for data and indices is set to 4,096 bytes. For the sake of fairness, we use the same index structure for each clustering criterion consistently: MBR-MLGF for the spatial index and the inverted index for the text index. We implemented all of the methods using the MBR-MLGF [20] and the inverted index that are part of the Odysseus DBMS [23, 24].

6.2 Results of the experiments

6.2.1 Index size

Table 7 shows the sizes of the indices. S_{M4} is bigger than S_{M123} by a factor of 1.83; S_{M5} is bigger than S_{M123} by 4.09. S_{M4} is bigger than S_{M123} due to the space of the inverted indices for internal nodes. S_{M5} is bigger than S_{M123} since the spatial index of S_{M123} linearly depends on N and that of S_{M5} linearly depends on $|W_T|$ where $|W_T| >> N$. This result supports the analysis of storage overhead in Section 5.2.

6.2.2 Query processing performance

Performance as $q.k$ is varied Figure 9a shows the query processing time and Figure 9b the number of page accesses as $q.k$ is varied. We find that the overall trends of the estimated results in Figure 7a are very close to the experimental results, and hence, we verify that the best methods estimated for the query loads with extreme values are consistent with those from the experimental results. Specifically, M2 is the best for a small $q.k$, and M1 is the best for a large $q.k$. For a large $q.k$, even if the number of page accesses of M1 is relatively larger, the query processing time of M1 is the smallest; this result is due to the effect of sequential access. For some middle values of $q.k$, the best methods are rather different between the estimated and experimental results. However, these differences are minor, and they are likely to be caused by the discrepancies between the actual data distribution and the uniform distribution we assumed for the cost model.

Table 6 Query parameters and values

Parameters	Values
$q.k$	1, **10**, 100, 1000
$q.p$	0.1, 0.3, **0.5**, 0.7, 0.9
$\lvert q.keywords \rvert$	1, 2, **3**, 4, 5

Table 7 Sizes of indices (MB)

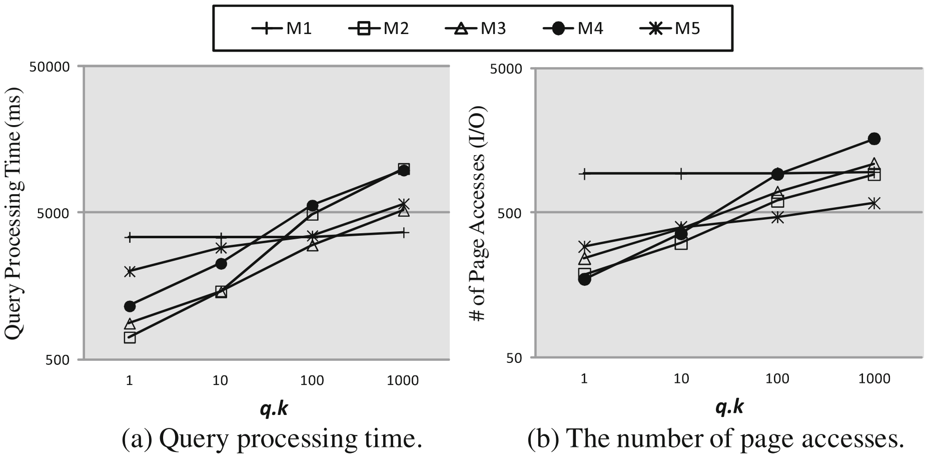
Method	*DataSet1*	*DataSet2*	*DataSet3*
M1/M2/M3	662	871	12,641
M4	979	1,594	22,788
M5	2,709	2,204	23,576

Performance as $q.p$ is varied Figure 10a shows the query processing time and Figure 10b the number of page accesses as $q.p$ is varied. We find that again the overall trends of the estimated results in Figure 7b are very close to the experimental results. Specifically, M4 and M5 show a better performance than M3 for a large $q.p$, and M3 is the best for a small $q.p$. Meanwhile, unlike the estimated result, the performance of M3 varies. However, its variation is relatively less than those of M4 and M5, and it is likely to be caused by the actual data distribution.

Performance as $|q.keywords|$ is varied Figure 11a shows the query processing time and Figure 11b the number of page accesses as $|q.keywords|$ is varied. We also find that the overall trends of the estimated results in Figure 7c are very close to the experimental results. Specifically, M5 is the best for a small $|q.keywords|$, and M3 is the best for a large $|q.keywords|$.

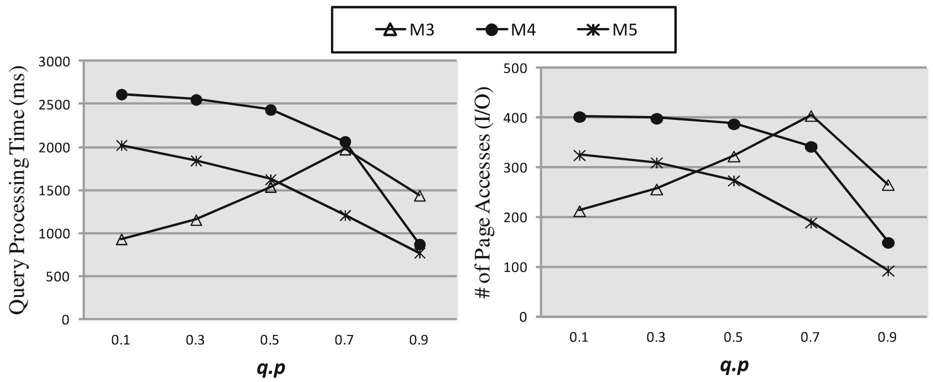
Performance as data set size is varied Figure 12 shows the performance as the data set size is varied. All the methods have similar trends as the data set size increases.

Experiments on DataSet2 We have performed the same extensive experiments on *DataSet2* as on *DataSet1*. Figure 13 shows the performance as $q.k$ is varied. The results are consistent with those for *DataSet1*. The experimental results for the other parameters are also consistent and so are omitted.



(a) Query processing time. (b) The number of page accesses.

Figure 9 Performance comparison on *DataSet1* as $q.k$ is varied ($q.p$=0.5 *and* $|q.keywords| = 3$)



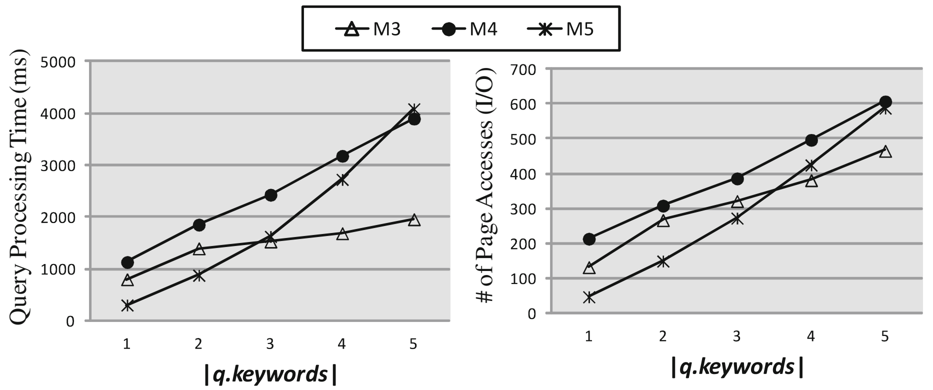
(a) Query processing time. (b) The number of page accesses.

Figure 10 Performance comparison on *DataSet1* as set as *q.p* is varied ($q.k = 10$ $and|q.keywords| = 3$)

Effect of buffering In the previous experiments, we have measured the performance under the environment removing the effect of buffering so as to assume the worst case. Now, to show the buffering effect, we compare the query performance of generated methods according to whether the buffer is used or not where the size of the buffer is 32MB. Figure 14 shows the reduction of the query performance due to buffering. The result shows that M3 and M4 take advantage of buffering since the same indices are used for every query; M5 cannot take advantage of buffering much since it maintains an index for each keyword and accesses different indices for the given query keywords.

7 Comparison with related work

IR-tree [5, 14] and S2I [17] are the representative methods for indexing top-k spatial-keyword queries. These methods are based on the combined clustering approach. IR-tree first builds the R-tree, and then, builds the inverted index on the objects in each leaf node



(a) Query processing time. (b) The number of page accesses.

Figure 11 Performance comparison on $DataSet1$ as $|q.keywords|$ is varied ($q.k = 10$ and $q.p = 0.5$)

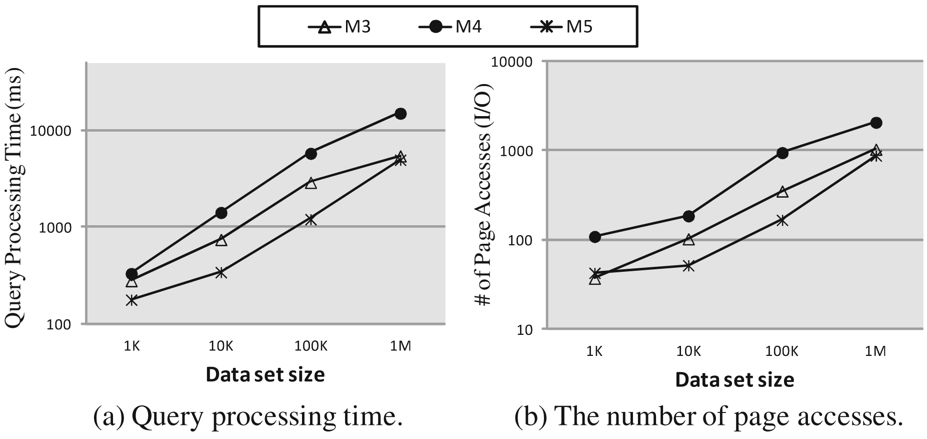


Figure 12 Performance comparison on as data set size is varied ($q.k = 10$, $q.p = 0.5$, $|q.keywords| = 3$)

of the R-tree. S2I first builds the inverted index, and then, builds the R-tree on the objects in each posting list of the inverted index. IR-tree is mapped to M4, and S2I is mapped to M5 as shown in Section 3.3. Recently, I^3 [25] and IL-Quadtree [26] have been proposed to support top-k spatial-keyword queries. In effect, I^3 and IL-Quadtree are equivalent to S2I except for using the quartree instead of the R-tree. RASIM [13] is the only method based on the separate clustering approach. It is mapped to M3 as shown in Section 3.3.

Cong et al. [5] have also proposed two variations of the IR-tree: 1) CIR-tree and 2) DIR-tree. First, CIR-tree improves the pruning power by partitioning the entire set of objects into a specified number of groups and by making an entry for each group in a child node instead of for each child node. Since this technique can be applied to any methods, it is orthogonal to the issues discussed in this paper. Next, DIR-tree partitions the entire set of objects into leaf nodes by considering not only location similarity but also textual similarity between objects. Here, the *system preference* is used for controlling the weight between location

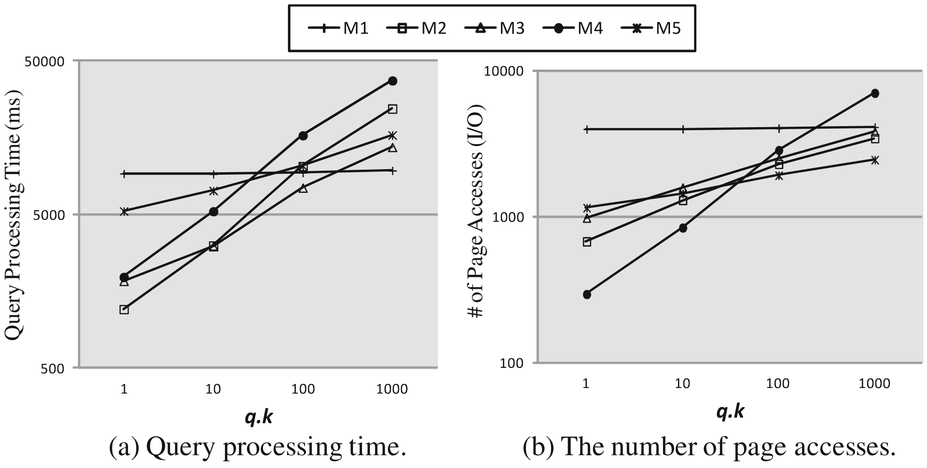
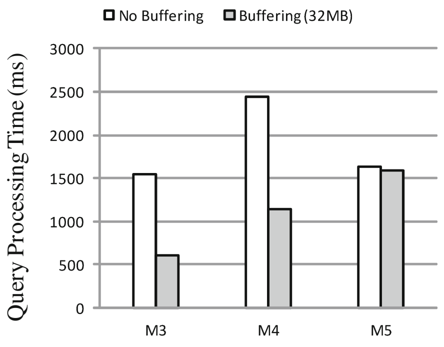
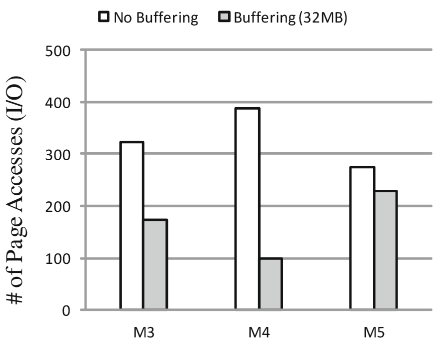


Figure 13 Performance comparison on $DataSet2$ as $q.k$ is varied ($q.p = 0.5 and |q.keywords| = 3$)



(a) Query processing time.



(b) The number of page accesses.

Figure 14 Performance comparison as data set size is varied ($q.k$ = 10, $q.p$ =0.5, and $|q.keywords|$ = 3)

similarity and textual similarity. We can map DIR-tree to a variation of M4 that considers not only location similarity but also textual similarity between objects when building the spatial index. However, the performance improvement of the method occurs only when the given user preference is similar to the system preference. Our experiments for M4 are equivalent to those for DIR-tree varying only the user preference while fixing the system preference to be 1.0 (i.e., considering only spatial proximity).

Many methods have been proposed for spatial-keyword queries without the top-*k* constraint. Chen et al. [3] have proposed a separate clustering method that supports efficient merging of the results retrieved from each index. This method can be mapped to SK-M1 in Section 4. There have been many combined clustering approaches that combine the spatial and text indices. Zhou et al. [27] and Vaid et al. [22] have proposed indices in which an inverted index is built on each leaf node of the R*-tree, or an R*-tree is built on each posting list of the inverted index. The former is can be mapped to SK-M3 in Section 4; the latter to SK-M4. Park et al. [15] have proposed an R-tree-based index combining S-tree, which is a hierarchical signature file with a structure-symmetric to the R-tree. Hariharan et al. [11] have proposed another R-tree-based index building an inverted index on each leaf node and storing a set of distinct keywords on each internal node. Felipe et al. [7] have proposed an other R-tree-based index augmenting each node of an R-tree with a signature file. These methods can be mapped to SK-M3 even if their text indices are different (i.e., a signature file or an inverted index). Christofaraki et al.[4] applies the recent text processing techniques to the combined clustering approach based on the inverted index. This method can be mapped to SK-M4.

8 Conclusions

We have proposed a generic model of index schemes for top-*k* spatial-keyword queries, called *G-Index Model*. G-Index Model is a unified framework for top-*k* spatial-keyword queries by exhaustively enumerating all the possible methods. For this, we have identified the clustering criteria and operators that can be used in the index schemes and have generated index schemes by combining them. According to this model, we have generated five methods: M1, M2, M3, M4, and M5. We have shown that all the existing methods are mapped to the generated methods, i.e., M3 (RASIM [13]), M4 (IR-tree [5, 14]), and

M5 (S2I [17]). This shows that our model sheds new light on understanding the existing methods under a unified framework. Using G-Index Model, we have also discovered new methods M1 and M2 that have not been reported before.

We have shown that G-Index Model can be applied into a class of queries involving arbitrary multiple data types. As concrete examples, we have shown that G-Index Model can generate index schemes for two classes of queries: the spatial-keyword query without the top-*k* constraint and the top-*k* spatial-keyword-relational query, which adds the relational type to the top-*k* spatial-keyword query. We have also constructed a cost model for estimating query processing cost of the generated methods for the top-*k* spatial-keyword query. Through an extensive experiment, we verify that the cost model is consistent with the experimental results. We note that the cost model can be used to do physical database design for supporting top-*k* spatial-keyword queries.

A very important observation on G-Index Model is that 1) G-Index Model is generic to deal with arbitrary multiple data types, 2) we provide a set of building blocks (i.e., clustering criteria and operators) and their combinations for generating index schemes, and 3) we prove their effectiveness through theoretical and empirical study.

Acknowledgments This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MEST) (No. 2012R1A2A1A05026326).

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 (1990)
3. Chen, Y., Suel, T., Markowetz, A.: Efficient query processing in geographic web search engines. In: Proceedings of the International Conference on Management of Data, ACM SIGMOD (2006)
4. Christoforaki, M., He, J., Dimopoulos, C., Markowetz, A., Suel, T.: Text vs. Space: Efficient geo-search query processing. In: Proceedings of the 20th ACM Conference on Information and Knowledge Management(CIKM) (2011)
5. 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) (2009)
6. 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) (2001)
7. Felipe, I., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: Proceedings of the 24th International Conference on Data Engineering (ICDE), IEEE (2008)
8. Finkel, R., Bentley, J.L.: Quad Trees: A data structure for retrieval on composite keys. Acta Informatica **4**(1), 1–9 (1974)
9. Garcia-Molina, H., Ullman, J., Widom, J., 2nd Ed. Database systems: The complete book. Prentice Hall, Englewood Cliffs (2008)
10. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proceedings of the International Conference on Management of Data, ACM SIGMOD (1984)
11. Hariharan, R., Hore, B., Li, C., Mehrotra, S.: Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) systems. In: Proceedings of the 19th International Conference on Scientific and Statistical Database Management (SSDBM) (2007)
12. Hjaltason, G., Samet, H.: Distance browsing in spatial databases. ACM Trans. Database Syst. **24**(2), 265–318 (1999)
13. 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)
14. 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)
15. Park, D., Kim, H.: An enhanced technique for k-nearest neighbor queries with non-spatial selection predicates. Multimed. Tools Appl. Arch. **19**(1), 79–103 (2003)

16. 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)
17. Rocha-Junior, J., Gkorgkas, O., Jonassen, S., Norvag, K.: Efficient processing of top-k spatial keyword queries. In: Proceedings of the 12th International Symposium on Spatial and Temporal Databases (SSTD) (2011)
18. Sanderson, M., Kohler, J.: Analyzing geographic queries. In: Proceedings of the 1st ACM SIGIR Workshop on Geographic Information Retrieval (2004)
19. 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) (1987)
20. Song, J., Whang, K., Lee, Y., Kim, S.: Spatial join processing using corner transformation. IEEE Trans. Knowl. Data Eng. **11**(4), 688–698 (1999)
21. 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)
22. Vaid, S., Jones, C., Joho, H., Sanderson, M.: Spatio-textual indexing for geographical. In: Proceedings of the 9th International Symposium on Spatial and Temporal Databases (SSTD) (2005)
23. 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, 5–8 April 2005. This paper received the Best Demonstration Award
24. 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)
25. Zhang, D., Tan, K., Tung, A.: Scalable top-k spatial keyword search. In: Proceedings of the 16th International Conference on Extending Database Technology(EDBT), ACM, 359–370 (2013)
26. 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, 901–912 (2013)
27. Zhou, Y., Xie, X., Wang, C., Gong, Y., Ma, W.: Hybrid index structures for location-based web search. In: Proceedings of the 14th ACM Conference on Information and Knowledge Management(CIKM), 155–162 (2005)