The VLDB Journal
DOI 10.1007/s00778-012-0271-0

REGULAR PAPER

A framework for efficient spatial web object retrieval

Dingming Wu · Gao Cong · Christian S. Jensen

Received: 20 April 2011 / Revised: 20 February 2012 / Accepted: 5 March 2012

Abstract The conventional Internet is acquiring a geospatial dimension. Web documents are being geo-tagged and geo-referenced objects such as points of interest are being associated with descriptive text documents. The resulting fusion of geo-location and documents enables new kinds of queries that take into account both location proximity and text relevancy. This paper proposes a new indexing framework for top-k spatial text retrieval. The framework leverages the inverted file for text retrieval and the R-tree for spatial proximity querying. Several indexing approaches are explored within this framework. The framework encompasses algorithms that utilize the proposed indexes for computing location-aware as well as region-aware top-k text retrieval queries, thus taking into account both text relevancy and spatial proximity to prune the search space. Results of empirical studies with an implementation of the framework demonstrate that the paper's proposal is capable of excellent performance.

Keywords Spatial web · Keyword query · Spatial query · Top-K query · Inverted file · R-tree · Spatio-textual indexing

D. Wu (✉)
Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong
e-mail: dmwu@comp.hkbu.edu.hk

G. Cong
School of Computer Engineering, Nanyang Technological University, Singapore, Singapore
e-mail: gaocong@ntu.edu.sg

C. S. Jensen
Department of Computer Science, Aarhus University, Aarhus, Denmark
e-mail: csj@cs.au.dk

1 Introduction

Driven in part by the emergence of the mobile Internet, the conventional Internet is acquiring a geo-spatial dimension. On the one hand, many (geo-referenced) points of interest—for example, stores, tourist attractions, hotels, entertainment services, public transport, and public services—are being associated with descriptive text documents. On the other hand, web documents are increasingly being geo-tagged.

This fusion of geo-location and documents enables queries that take into account both spatial proximity and text relevancy. One study has found that about one fifth of web search queries are geographical and have local intent, as determined by the presence of geographical terms such as place names and postal codes [31]. Indeed, commercial search engines have started to provide location-based services, such as map services, local search, and local advertisements. For example, Google Maps supports location-aware text retrieval queries. Additional examples of location-based services include online yellow pages.

This paper proposes a kind of top-k query that takes into account both spatial proximity and text relevancy for points of interest with associated text. An example query may request a "good micro-brewery that serves pizza" and that is close to the user's hotel. We call this type of query a *top-k spatial text retrieval* query. It consists of a spatial component (the user's hotel) and a text component (good micro-brewery that serves pizza). This kind of query is different from the query that retrieves relevant documents within a geographical range.

More specifically, we consider two types of *top-k spatial text retrieval* queries that differ in their spatial components. The *Location-aware top-k Text retrieval* (LkT) query has a point location as its spatial component. The answer to the LkT query is a list of k objects ranked according to a ranking

Published online: 18 March 2012

function that combines their distances to the query location and the relevance of their textual descriptions to the query text. The *Region-aware top-k Text retrieval* (R*k*T) query has a region (e.g., a rectangle) as its spatial component. We consider two variants of the R*k*T query that yield different results. The motivation underlying the R*k*T queries is that the spatial component of a query may be inaccurate and thus is better modeled by a region rather than a point location. This inaccuracy may be due to several reasons. First, positioning devices might not be accurate enough to provide point locations. Second, in some applications, for example, Google Maps, users enter keywords to represent their locations, for example, the names of streets, which are regions. Third, for privacy reasons, users may not want to reveal their accurate locations by enlarging points to regions.

In the paper, we compute the text relevancy of a query result by means of language models and a probabilistic ranking function that have sound foundations in statistical theory and that have performed well empirically in many information retrieval tasks [28,36].

The *top-k spatial text retrieval* queries pose new challenges to both existing spatial database and existing information retrieval techniques that have evolved separately. The research in spatial databases mainly focuses on highly structured, map-based geometric data and their attributes. In contrast, information retrieval research often treats location information as common keywords.

We propose a new indexing framework for processing *top-k spatial text retrieval* queries. This framework integrates the inverted file for text retrieval and the R-tree for spatial proximity querying to obtain an Inverted file R-tree, called the IR-tree, that is essentially an R-tree extended with inverted files. Associated algorithms are proposed for the processing of the L*k*T and R*k*T queries that are capable of pruning the search space by simultaneously making use of both spatial proximity and text relevancy.

Each node of the IR-tree records a summary of the location information and the textual content of all the objects in the subtree rooted at the node. The query processing algorithms utilize the location information to estimate the spatial distance of a query to the objects in the node's subtree, and they use the text information to estimate the text relevancy scores for these objects.

We also explore a variant of the IR-tree that incorporates document similarity when computing *Minimum Bounding Rectangles* (MBRs), yielding a new index called the DIR-tree. While the IR-tree considers only location information when generating its MBRs, the DIR-tree takes into account both location information and document similarity. The IR-tree can be seen as a special case of the DIR-tree. To further improve the performance of the IR-tree and the DIR-tree, we cluster the documents attached to spatial objects. In an index node, tighter text relevancy scores can be estimated for a group of similar documents than for diverse documents that belong to different categories. This cluster-enhanced method can be applied to both the IR-tree and the DIR-tree, yielding the CIR-tree and the CDIR-tree.

To further optimize the IR-tree, we propose two techniques to derive tighter MBRs. The first technique constructs an MBR for each term (word) in the inverted files of each node, which we call a TermMBR. This technique can be applied to all the above-mentioned four indexes. The second technique constructs an MBR for each cluster of each node in the CIR-tree and the CDIR-tree, which we call a ClusterMBR.

The paper extends a previously published conference paper [8]. Beyond a recent paper [22] that proposes a variation of the IR-tree presented in the conference paper, we are not aware of any techniques in the literature that efficiently support the computation of the top-*k* spatial text retrieval queries considered in this paper. In Sect. 8, we cover the extensions over these papers in detail. Some techniques [24,34,38] use an ad hoc combination of nearest neighbor (NN) and keyword search techniques for *spatial text retrieval*. For example, an R-tree is used to find the nearest neighbors, and then for each neighbor, an inverted file is used to rank the objects according to text relevancy. This ad hoc combination cannot easily be applied to process *top-k spatial text retrieval* queries because it is difficult to determine in advance the number of nearest neighbors needed to obtain the top-*k* results ranked by a combination of spatial proximity and text relevancy. A recent proposal [10] integrates the R-tree with signature files. However, this proposal is inapplicable to *top-k spatial text retrieval* (which involves ranking, not just the checking of Boolean predicates) mainly due to the use of signature files, which cannot sensibly handle ranked text retrieval [39].

In summary, the paper's contribution is threefold. First, we introduce two types of *top-k spatial text retrieval* queries, called L*k*T and R*k*T queries, that return objects ranked according to a linear interpolation function that combines normalized spatial proximity and text relevancy.

Second, to efficiently process these queries, we propose a new indexing framework that integrates location indexing and text indexing. Specifically, we develop the IR-tree and associated algorithms for the processing of L*k*T and R*k*T queries. A variant of the IR-tree, the DIR-tree, is proposed to incorporate document similarity when computing MBRs. We also exploit document clustering to improve the indexing framework. Additionally, we propose so-called ClusterMBRs and TermMBRs to compute tighter MBRs with the aim of improving query processing performance.

Third, we evaluate the paper's proposals. Results of empirical studies with implementations of the proposed techniques demonstrate that the paper's proposals offer scalability and

are capable of excellent performance. An analytical study offers insight into the query performance of the IR-tree and suggests that it outperforms the recent variant [22] quite significantly in realistic settings.

The rest of this paper is organized as follows. Section 2 formally defines the Location-aware top-k Text retrieval problem and the Region-aware top-k Text retrieval problem. Two baseline algorithms are proposed in Sect. 3. Section 4 presents the index framework for processing the LkT query. Section 5 proposes four methods for enhancing the framework. Algorithms for RkT queries are presented in Sect. 6. We cover the empirical performance study in Sect. 7. Finally, we cover related work in Sect. 8 and offer conclusions and research directions in Sect. 9.

2 Problem statement

Let $\mathscr{D}$ be a database. Each spatial web object O in $\mathscr{D}$ is defined as a pair $(O.\lambda, O.\psi)$, where $O.\lambda$ is a location descriptor in multidimensional space and $O.\psi$ is a document (e.g., a dining menu) that describes the object (e.g., an Italian restaurant). We assume a two-dimensional geographical space composed of latitude and longitude, but the paper's proposals generalize to other multidimensional spaces of low dimensionality.

Intuitively, a *Location-aware top-k Text retrieval* (LkT) query retrieves k objects in database $\mathscr{D}$ for a given query Q such that their locations are the closest to the location specified in Q and their textual descriptions are the most relevant to the keywords in Q. Formally, given a query Q, defined as a pair $(Q.\lambda, Q.\psi)$, where $Q.\lambda$ is a location descriptor and $Q.\psi$ is a set of keywords, the objects returned are ranked according to a ranking function given by: $f(D_\varepsilon(Q.\lambda, O.\lambda), P(Q.\psi|\ O.\psi))$, where $D_\varepsilon(Q.\lambda, O.\lambda)$ is the Euclidian distance between Q and O and $P(Q.\psi|O.\psi)$ is the text relevancy of $O.\psi$ with regard to $Q.\psi$. The text relevancy can be computed as the probability of generating query $Q.\psi$ from the language models of the documents or other text models.

We tackle the problem of efficiently answering LkT queries. Thus, given a query Q, we retrieve a ranked list of k objects according to their ranking scores as computed by the ranking function $f(\cdot,\cdot)$ introduced above. The paper's proposals are applicable to a wide range of ranking functions, namely all functions that are monotone with respect to distance proximity and text relevancy.

In this paper, we follow existing work and use linear combinations [24]. Specifically, we derive a ranking function as a weighted sum of normalized terms for ranking an object O with regard to a query Q, called *spatial-textual distance* and denoted by $D_{ST}(Q, O)$.

$$D_{ST}(Q, O) = \alpha \frac{D_\varepsilon(Q.\lambda, O.\lambda)}{maxD} + (1-\alpha)\left(1 - \frac{P(Q.\psi|O.\psi)}{maxP}\right), \quad (1)$$

where $\alpha \in [0, 1]$ is a parameter used to balance spatial proximity and text relevancy; the Euclidian distance between Q and O, $D_\varepsilon(Q.\lambda, O.\lambda)$, is normalized by $maxD$, which can be, for example, the maximum distance between two objects in $\mathscr{D}$; and $maxP$ is used to normalize the text relevancy score into the range from 0 to 1. Note that the lower the score computed by ranking function, the better.

The parameter α in Eq. 1 allows users to set their preferences between text relevancy and location proximity at query time. Note that this study focuses on efficient solutions, not on new effective ranking functions.

The query location $Q.\lambda$ in an LkT query is a point. In some applications, as stated in Sect. 1, users may give a region rather than an accurate point location. A *Region-aware top-k Text retrieval* (RkT) query Q, defined as a pair $(Q.\Omega, Q.\psi)$ covers this scenario, where $Q.\Omega$ is a region, for example, a rectangle. We propose two specializations of the RkT query: RkT^e and RkT^u.

The RkT^e query Q returns exactly k objects and uses the same ranking function as does the LkT query. The only difference is that if $O.\lambda \in Q.\Omega$ then $D_\varepsilon(Q.\Omega, O.\lambda) = 0$; otherwise, the Euclidean distance $D_\varepsilon(Q.\Omega, O.\lambda)$ is defined as the minimum Euclidian distance between Q and O, denoted by $mindist(Q.\Omega, O.\lambda)$.

The result of the RkT^u query Q is not a ranked list, but the union of the $LkT(q, Q.\psi)$ result sets, where q is every possible point in region $Q.\Omega$. Formally,

$$RkT^u(Q.\Omega, Q.\psi) = \bigcup_{q \in Q.\Omega} \{LkT(q, Q.\psi)\}. \quad (2)$$

In the definition, we use set notation ("$\{\cdots\}$") to emphasize that the LkT query results are viewed as sets. Thus, the result of the RkT^u query is a union of sets, which is a set.

The RkT^e and the RkT^u queries differ in how distances between objects and queries are measured. In general, the RkT query targets application scenarios where users cannot provide accurate point locations, for example, due to inaccurate positioning or the service provider not having access to the users' locations due to privacy concerns. If we assume that the query point location is hidden inside the spatial region given in the query and the real result is known, the RkT^e and the RkT^u queries both return approximate results. The RkT^e query considers the minimum Euclidean distance between the query region and the objects, so its result may contain some false hits and may miss some true result objects. The result of the RkT^u query is a superset of the real result, since it

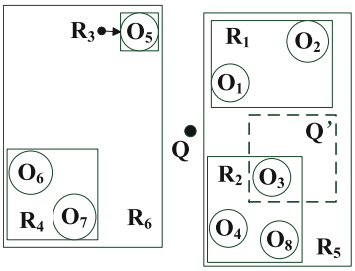

Fig. 1 Objects and their bounding rectangles

Table 1 Document-by-term matrix

	Chinese	Spanish	Restaurant	Food
$O_1.\psi$	0.5	–	0.5	–
$O_2.\psi$	–	0.5	0.5	–
$O_3.\psi$	0.7	–	–	0.1
$O_4.\psi$	–	–	0.7	0.1
$O_5.\psi$	0.4	–	0.4	–
$O_6.\psi$	–	0.4	0.3	–
$O_7.\psi$	0.1	0.1	0.4	0.1
$O_8.\psi$	–	0.3	0.3	–

Table 2 Distances between queries and objects

	$D_\varepsilon(Q.\lambda,)$	$mindist(Q'.\Omega,)$	$maxdist(Q'.\Omega,)$
$O_1.\lambda$	0.2	0.1	0.3
$O_2.\lambda$	0.5	0.2	0.4
$O_3.\lambda$	0.6	0	0.2
$O_4.\lambda$	0.7	0.1	0.3
$O_5.\lambda$	0.3	0.3	0.5
$O_6.\lambda$	0.9	0.8	0.9
$O_7.\lambda$	0.8	0.6	0.8
$O_8.\lambda$	0.8	0.1	0.2

unions the results of all possible query point locations inside the query region.

Example 1 Figure 1 shows 8 spatial web objects $O_1, \ldots, O_8$, and Table 1 shows a document-by-term matrix of their documents. For example, the matrix shows that the weight of term *Chinese* in document $O_1.\psi$ is 0.5. The weight of a term that does not appear in a document is set to a small value, for example, 0.001. Given an LkT ($k = 1$) query Q with location $Q.\lambda$ as shown in Fig. 1 and $Q.\psi =$ (Chinese restaurant), object O_1 is the result with ranking score 0.475 according to Tables 1 and 2 ($\alpha = 0.5$). Given an RkT^e ($k = 1$) query Q' with region $Q'.\Omega$, shown as a dashed rectangle in Fig. 1, and $Q'.\psi =$ (Spanish food), object O_3 is the result with ranking score 0.49995 according to Tables 1 and 2 ($\alpha = 0.5$). If Q' is an RkT^u ($k = 1$) query, the results are O_2, O_3, O_4, O_8.

3 Baselines

We next discuss how to exploit existing techniques for processing LkT queries. While no baseline algorithm exists for LkT queries, a straightforward baseline is to adapt an existing approach [24], thus computing the text relevancy using an inverted file and computing location proximity using an R-tree separately for all objects and then combining them to obtain the top-k objects. This is not efficient. The main difficulty is in benefitting from both the inverted file and the R-tree.

The two new baseline algorithms presented next: Inverted File Only (IFO) and R-tree plus Inverted File (RIF) are inspired by the Threshold algorithm [12].

Baseline 1: IFO. The idea is to utilize the inverted file to compute the text relevancy scores of all objects (corresponding to the right operand of the operator "+" in Eq. 1), thus obtaining a list *IRRanking* that ranks objects in ascending order of their scores. The list is then scanned to compute the spatial proximity to the query until further scanning will not generate top-k results. Only the inverted file is used.

During a scan, the algorithm keeps track of the combined ranking score (defined in Eq. 1; the lower the score, the better) of the current k'th object, denoted by *threshold*. If the *IRRanking* score of a new object T exceeds *threshold*, the algorithm stops since all objects after T in *IRRanking* also have a score that exceeds *threshold*; otherwise, we retrieve its location, compute its combined ranking score, and compare with *threshold* to determine whether *threshold* needs to be updated.

Example 2 Table 3 illustrate the use of IFO. Consider an LkT ($k = 3$) query on a dataset of 9 objects. Initially (Step 1), the *IRRanking* list is computed using the inverted file and the *threshold* is set to ∞. Next (Step 2), the first object O_9 in the *IRRanking* list is removed and added to the *Candidates* list with its combined ranking score. Then, we repeatedly remove the first object in the *IRRanking* list and add it to the *Candidates* list with its combined score. The objects in the *Candidates* list occur in ascending order of their combined scores, and *threshold* always points to the third object in the list. When the score of the first object in *IRRanking* eventually exceeds *threshold*, the algorithm stops, and the top three objects O_7, O_2, O_9 in *Candidates* are returned (Step 6).

Baseline 2: RIF. This algorithm uses an R-tree and an inverted file in two stages. The inverted file is used for computing the list *IRRanking* as for IFO. The algorithm then incrementally finds nearest neighbors [18] using the R-tree and checks the text relevancy scores of objects in *IRRanking*.

In the process, the algorithm keeps track of the minimum text relevancy score in *IRRanking*, denoted by *MinTR*, that

Table 3 Example of IFO

Step 1		Step 2		Step 3		...	Step 6	
IR ranking	Candidates	IR ranking	Candidates	IR ranking	Candidates	...	IR ranking	Candidates
O_9, 0.25	∞	O_7, 0.31	O_9, 0.5	O_2, 0.42	O_7, 0.4	...	O_6, 0.51	O_7, 0.4
O_7, 0.31	∞	O_2, 0.42	∞	O_3, 0.44	O_9, 0.5	...	O_4, 0.53	O_2, 0.48
O_2, 0.42	$\boxed{\infty}$	O_3, 0.44	$\boxed{\infty}$	O_1, 0.48	$\boxed{\infty}$	...	O_8, 0.56	$\boxed{O_9, 0.5}$
O_3, 0.44		O_1, 0.48		O_6, 0.51		...	O_5, 0.62	O_1, 0.51
O_1, 0.48		O_6, 0.51		O_4, 0.53		...		O_3, 0.52
O_6, 0.51		O_4, 0.53		O_8, 0.56		...		
O_4, 0.53		O_8, 0.56		O_5, 0.62		...		
O_8, 0.56		O_5, 0.62				...		
O_5, 0.62						...		

Boxed: threshold

Table 4 Example of RIF

Step 1		Step 2		Step 3		...	Step 6	
IR ranking	Candidates	IR ranking	Candidates	IR ranking	Candidates	...	IR ranking	Candidates
O_9, $\underline{0.25}$	∞	O_9, $\underline{0.25}$	O_1, 0.51	O_9, $\underline{0.25}$	O_2, 0.48	...	O_6, $\underline{0.51}$	O_7, 0.4
O_7, 0.31	∞	O_7, 0.31	∞	O_7, 0.31	O_1, 0.51	...	O_4, 0.53	O_2, 0.48
O_2, 0.42	$\boxed{\infty}$	O_2, 0.42	$\boxed{\infty}$	O_3, 0.44	$\boxed{\infty}$	...	O_8, 0.56	$\boxed{O_9, 0.5}$
O_3, 0.44		O_3, 0.44		O_6, 0.51		...	O_5, 0.62	O_1, 0.51
O_1, 0.48		O_6, 0.51		O_4, 0.53		...		O_3, 0.52
O_6, 0.51		O_4, 0.53		O_8, 0.56		...		
O_4, 0.53		O_8, 0.56		O_5, 0.62		...		
O_8, 0.56		O_5, 0.62				...		
O_5, 0.62						...		

Boxed: threshold; underlined: MinTR

has not been "seen" so far, and the combined ranking score of the current kth object, denoted by *threshold*.

For a newly "seen" object with spatial distance *dist*, if the combined score computed from *dist* and *MinTR* exceeds *threshold*, the algorithm stops since it is guaranteed that all "unseen" objects will not have lower scores than the current kth object (and thus cannot be in the result).

Example 3 Table 4 illustrates the RIF algorithm. Consider an LkT ($k = 3$) query on the 9 objects from Example 2. Initially (Step 1), the *IRRanking* list is computed using the inverted file, the *threshold* is set to ∞, and *MinTR* is set to 0.25. Next (Step 2), the R-tree reports the nearest object O_1 with $dist = 0.03$. Since the combined score computed from *dist* and the current *MinTR* is smaller than *threshold*, object O_1 is removed from the *IRRanking* list and added to the *Candidates* list with its combined score computed from $dist = 0.03$ and its text relevancy score (i.e., 0.48) in the *IRRanking* list. Then, the R-tree continuously reports the next nearest object, and the processing in Step 2 is repeated. The objects in the *Candidates* list are ordered ascendingly on their combined scores, and *threshold* always points to the third object in the list. Also, *MinTR* always points to the first object in the *IRRanking* list. After several steps, when the R-tree reports an object (e.g., O_4) such that the combined score computed from its *dist* and the current *MinTR* exceeds *threshold*, the algorithm stops, and the top three objects O_7, O_2, O_9 in *Candidates* are returned (Step 6).

4 Hybrid indexing for location-aware text retrieval

We present a framework that integrates the R-tree and the inverted file into a new index, the *Inverted file R-tree* (IR-tree), and that includes an algorithm for processing LkT queries using the IR-tree.

4.1 Hybrid index framework: the IR-tree

The R-tree [16] is arguably the dominant index for spatial queries, and the inverted file is the most efficient index for

information retrieval [39]. These were developed separately and for different kinds of queries.

We aim to develop an approach that is able to leverage both techniques for the efficient processing of LkT queries. To achieve this goal, a simple approach is to use the inverted file (resp. the R-tree) to generate a number of top candidate objects based on text relevancy (resp. spatial proximity) and then compute the spatial distances (resp. text relevancy) of the candidate objects using the other index. However, this approach is not efficient since there is no sensible way to determine the number of candidate objects needed from the first step in order to ensure that the top-k objects are found in the end. Instead, we propose a hybrid indexing structure, the IR-tree, that utilizes both indexing structures in a combined fashion.

The IR-tree is essentially an R-tree, each node of which is enriched with a reference to an inverted file for the objects contained in the subtree rooted at the node.

In the IR-tree, a leaf node contains a number of entries of the form $(O, O.r)$, where O refers to an object in database $\mathscr{D}$ and $O.r$ is the bounding rectangle of object O. A leaf node also contains a pointer to an inverted file for the text documents of the objects being indexed. The inverted file is stored separately from the R-tree, for two reasons: First, it is more efficient to store each inverted file contiguously, rather than as a sequence of blocks or pages that are scattered across a disk [39]. Second, the inverted file can be distributed across several machines, while this is not easily possible for the R-tree [32].

An inverted file consists of the following two main components.

- A vocabulary of all distinct terms in a collection of documents.
- A set of posting lists, each of which relates to a term t. Each posting list is a sequence of pairs $\langle O, w\rangle$, where O refers to an object whose document $O.\psi$ contains term t, and w is the weight of term t in document $O.\psi$.

A non-leaf node N contains a number of entries of the form $(e, e.r)$ where e points to a child node of N and $e.r$ is the Minimum Bounding Rectangle (MBR) of all rectangles in entries of the child node. A pseudo document is constructed for each non-leaf entry in the IR-tree.

The pseudo document is an important concept in the IR-tree. It represents all documents in the entries of the child node, enabling us to estimate a bound on the text relevancy to a query of all documents contained in the subtree rooted at e. The weight of a term t in the pseudo document referenced by e is the maximum weight of t in the documents contained in the subtree rooted at node e. A non-leaf node N also contains a pointer to an inverted file for the pseudo documents of the entries stored in N.

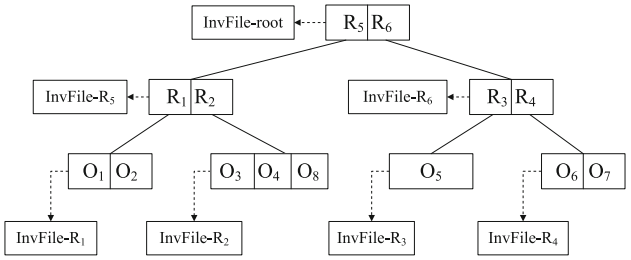

Fig. 2 Hybrid index IR-tree

Table 5 Contents of inverted files

	Chinese	Spanish	Restaurant	Food
InvFile-R$_1$	$\langle O_1, 0.5\rangle$	$\langle O_2, 0.5\rangle$	$\langle O_1, 0.5\rangle$	
			$\langle O_2, 0.5\rangle$	
InvFile-R$_2$	$\langle O_3, 0.7\rangle$	$\langle O_8, 0.3\rangle$	$\langle O_4, 0.7\rangle$	$\langle O_3, 0.1\rangle$
			$\langle O_8, 0.3\rangle$	$\langle O_4, 0.1\rangle$
InvFile-R$_3$	$\langle O_5, 0.4\rangle$		$\langle O_5, 0.4\rangle$	
InvFile-R$_4$	$\langle O_7, 0.1\rangle$	$\langle O_6, 0.4\rangle$	$\langle O_6, 0.3\rangle$	$\langle O_7, 0.1\rangle$
		$\langle O_7, 0.1\rangle$	$\langle O_7, 0.4\rangle$	
InvFile-R$_5$	$\langle R_1, 0.5\rangle$	$\langle R_1, 0.5\rangle$	$\langle R_1, 0.5\rangle$	$\langle R_2, 0.1\rangle$
	$\langle R_2, 0.7\rangle$	$\langle R_2, 0.3\rangle$	$\langle R_2, 0.7\rangle$	
InvFile-R$_6$	$\langle R_3, 0.4\rangle$	$\langle R_4, 0.4\rangle$	$\langle R_3, 0.4\rangle$	$\langle R_4, 0.1\rangle$
	$\langle R_4, 0.1\rangle$		$\langle R_4, 0.4\rangle$	
InvFile-root	$\langle R_5, 0.7\rangle$	$\langle R_5, 0.5\rangle$	$\langle R_5, 0.7\rangle$	$\langle R_5, 0.1\rangle$
	$\langle R_6, 0.4\rangle$	$\langle R_6, 0.4\rangle$	$\langle R_6, 0.4\rangle$	$\langle R_6, 0.1\rangle$

Example 4 Figure 2 illustrates the IR-tree for the 8 objects in Fig. 1. Table 5 shows the inverted files of the nodes in the IR-tree (Fig. 2). As a specific example, the weight of the term *restaurant* in entry R_2 in InvFile-R$_5$ is 0.7, which is the maximum weight of the term in the three documents O_3, O_4, O_8 in node R_2. Term t has the same weight in all the documents that do not contain t, which is determined by the maximum likelihood estimate of t in the whole dataset. These values are stored separately.

We proceed to present an important metric, the *minimum spatial-textual distance*, denoted by $MIND_{ST}(\cdot, \cdot)$, which will be used in the query processing. Given a query Q and a node N in the IR-tree, the metric $MIND_{ST}(Q, N)$ offers a lower bound on the actual spatial-textual distance $D_{ST}(\cdot, \cdot)$ between query Q and the objects enclosed in the rectangle of node N. This bound can be used to order and efficiently prune the search space in the hybrid index.

Definition 1 The *minimum spatial-textual distance* of a query Q from a node N in the hybrid index, denoted as $MIND_{ST}(Q, N)$, is defined as follows:

$$MIND_{ST}(Q, N) = \alpha \frac{mindist(Q.\lambda, N.r)}{maxD} + (1-\alpha)\left(1 - \frac{P(Q.\psi|N.\psi)}{maxP}\right), \quad (3)$$

where α, $maxD$, and $maxP$ are the same as in Eq. 1; $P(Q.\psi|N.\psi)$ is the text relevancy of the pseudo document of node N; and $mindist(Q.\lambda, N.r)$ is the minimum Euclidian distance between $Q.\lambda$ and $N.r$ (the MBR of N).

A salient feature of the proposed hybrid indexing structure is that it inherits the nice properties of the R-tree for query processing.

Theorem 1 *Given a query Q and a node N whose rectangle encloses a set of objects $SO = \{O_i, 1 \le i \le m\}$, the following is true: $\forall O_i \in SO\ (MIND_{ST}(Q, N) \le D_{ST}(Q, O_i))$.*

Proof Since object O_i is enclosed in the rectangle of node N, the minimum Euclidian distance between $Q.\lambda$ and $N.r$ is no larger than the Euclidian distance between $Q.\lambda$ and $O_i.\lambda$, that is, $mindist(Q.\lambda, N.r) \le D_\varepsilon(Q.\lambda, O_i.\lambda)$.

For each term t, its weight in $N.\psi$ (the pseudo document of node N) is the maximum weight of t in all the documents in node N. Thus, $P(Q.\psi|N.\psi) \ge P(Q.\psi|O_i.\psi)$.

According to Eqs. 1 and 3, we have $MIND_{ST}(Q, N) \le D_{ST}(Q, O_i)$, thus completing the proof. □

When searching the hybrid index for the top-k objects given a query Q, one must decide at each visited node of the hybrid index which entry to search first. Metric $MIND_{ST}$ offers an approximation of the spatial-textual distance to every entry in the node and, therefore, can be used to direct the search. Note that only the posting lists of keywords in query Q, but not all posting lists, are loaded into memory at a node to compute $MIND_{ST}$.

We next present an algorithm for building the IR-tree. The IR-tree is constructed by means of an insert operation that is adapted from the corresponding R-tree operation [16]. Algorithm 1 shows the Insert algorithm. It takes two arguments, the *MBR* and the *document* of an object. It uses a standard implementation of the R-tree [16] with operations ChooseLeaf and Split.

We may characterize the additional disk storage required by the IR-tree by comparing with the original R-tree and the inverted file. The number of nodes in the IR-tree is the same as that of the original R-tree, and the size of the inverted files contained in all leaf nodes of the IR-tree is comparable with that of the original inverted file. The IR-tree needs additional space to store the inverted files in its non-leaf nodes, the sizes of which depend on the number of non-leaf nodes and the storage utilization of nodes. If the capacity of each node is 100 entries, the length of the posting list for one word is at most 100 in a non-leaf node, which is independent of the number of objects contained in the subtree rooted at the non-lead node. The size of the inverted file at a non-leaf node is thus much smaller than that of the original inverted file. The paper's experimental study covers storage space.

Algorithm 1 Insert(*MBR*, *document*)

```
1: N ← ChooseLeaf(MBR);
2: Add MBR to node N, add document to the inverted file of N;
3: if N needs to be split then;
4:    {N1, N2} ← N.Split();
5:    if N is root node then
6:       Initialize a new node M;
7:       Add N1 and N2 to node M and update the inverted file of
   N1, N2 and M;
8:       Set M to the root node;
9:    else
10:      Ascend from N to the root, adjusting covering rectangles,
   updating the inverted file and propagating node splits as necessary;
11: else if N is not root then
12:    Update the covering rectangles and inverted files of the ancestor
   nodes of N;
```

In our implementation, we use a single dictionary for the inverted files of all nodes. Posting lists are indexed in B$^+$-trees. Each word is mapped to an ID and is used as the key of the B$^+$-trees.

4.2 Processing of LkT queries

To process LkT queries with the hybrid index framework, we exploit the best-first traversal algorithm (e.g., [18]) for retrieving the top-k objects. With the best-first traversal algorithm, a priority queue is used to keep track of the nodes and objects that have yet to be visited. The values of $D_{ST}(\cdot, \cdot)$ and $MIND_{ST}(\cdot, \cdot)$ are used as the keys of objects and nodes, respectively.

When deciding which node to visit next, the algorithm picks the node N with the smallest $MIND_{ST}(Q, N)$ value in the set of all nodes that have yet to be visited. The algorithm terminates when k objects (ranked according to Eq. 1) have been found. Algorithm 2 shows the pseudo-code.

We proceed to explain the algorithm and the use of the priority queue in the algorithm with an example.

Example 5 Consider the query Q ($Q.\psi$ = (Chinese restaurant) in Fig. 1. We want to find the top-1 object. We give the spatial-textual distances $D_{ST}(\cdot, \cdot)$ (Eq. 1) between query Q and all objects, as well as $MIND_{ST}(\cdot, \cdot)$ (Eq. 3) and the minimum Euclidian distances between Q and all bounding rectangles in Table 6 ($\alpha = 0.5$). Note that Algorithm LkT only computes the distances between Q and the objects or rectangles traversed by the algorithm, not all the distance in Table 6.

The algorithm uses a priority queue that contains the objects and bounding rectangles listed together with their D_{ST} and $MIND_{ST}$ scores, in increasing order of the scores, with ties broken by the alphabetical ordering. The algorithm

Algorithm 2 LkT(*Query*, *Index*, k)

```
1: Queue ← NewPriorityQueue();
2: Queue.Enqueue(Index.RootNode, 0);
3: while not Queue.IsEmpty() do
4:     Element ← Queue.Dequeue();
5:     if Element is an object then
6:         if not Queue.IsEmpty() and
7: D_ST(Query, Object) > Queue.First().Key then
8:             Queue.Enqueue(Object, D_ST(Query, Object));
9:         else
10:            Report Element as the next result object;
11:            if top-k objects have been found then
12:                break;
13:    else if Element is a leaf node then
14:        for each entry(Object) in leaf node Element do
15:            Queue.Enqueue(Object, D_ST(Query, Object));
16:    else
17:        for each entry(Node) in node Element do
18:            Queue.Enqueue(Node, MIND_ST(Query, Node));
```

Table 6 Spatial-textual distances

	$D_{ST}(Q, O_i)$		$mindist(Q.\lambda, R_i.r)$	$MIND_{ST}(Q, R_i)$
O_1	0.475	R_1	0.2	0.475
O_2	0.74975	R_2	0.2	0.355
O_3	0.79965	R_3	0.4	0.62
O_4	0.84965	R_4	0.5	0.73
O_5	0.57	R_5	0.05	0.28
O_6	0.94985	R_6	0.15	0.495
O_7	0.88			
O_8	0.89985			

starts by enqueueing the entries in the root and then executes the following steps:

(1) Queue:{(R_5,0.28),(R_6,0.495)}
Dequeue R_5, enqueue R_1 and R_2.
(2) Queue:{(R_2,0.355),(R_1,0.475),(R_6,0.495)}
Dequeue R_2, enqueue O_3, O_4, and O_8.
(3) Queue:{(R_1,0.475),(R_6,0.495),(O_3,0.79965),
(O_4,0.84965),(O_8,0.89985)}
Dequeue R_1, enqueue O_1 and O_2.
(4) Queue:{(O_1,0.475),(R_6,0.495),(O_3,0.79965),
(O_2,0.74975),(O_4,0.84965),(O_8,0.89985)}
Dequeue O_1. Report O_1 as the top-1 object. Terminate.

Observe that in the example, the algorithm does not traverse the entire tree in Fig. 2 since the search space is being pruned. However, the algorithm still visits some nodes that contain no results. Before reporting O_1, nodes *root*, R_5, R_2, and R_1 are visited. Since O_1 is the top-1 object, ideally we would visit *root*, R_5, and R_1, but not R_2. The reason why R_2 is visited is that the value $MIND_{ST}$ of R_2 is less than that of R_1.

In Table 5, we can see that objects O_3 and O_4 are very different. The weight of term *Chinese* is 0.7 in $O_3.\psi$, but $O_4.\psi$ does not contain it. On the other hand, the weight of term *restaurant* is 0.7 in $O_4.\psi$, but $O_3.\psi$ does not contain it. Since R_2 contains the two objects, the weights of *Chinese* and *restaurant* in $R_2.\psi$ are both 0.7. The reason for R_2 being highly relevant to the query is that it mixes objects of different types such that the weights of many terms are high in the pseudo document for R_2.

Based on this observation, we proceed to discuss how to enhance the query processing performance offered by the framework.

5 Enhanced hybrid indexing

We extend the hybrid indexing framework by incorporating document similarity, yielding an index called the DIR-tree (Document similarity enhanced Inverted file R-tree) in Sect. 5.1. The IR-tree can be viewed as a special case of the DIR-tree. Section 5.2 presents a clustering enhanced method for improving the indexing framework. We present the ClusterMBR enhancement in Sect. 5.3 and the Term-MBR enhancement in Sect. 5.4. We briefly discuss updates in Sect. 5.5.

5.1 Incorporating document similarity

Like the R-tree, the IR-tree is built based on the heuristic of minimizing the area of each enclosing rectangle in the inner nodes. Thus, the tree aims to place nodes that are spatially close in the same higher-level node. However, the spatial objects considered in this paper also have associated documents, and the LkT query takes into account both location proximity and text relevancy.

Unlike the IR-tree, the DIR-tree aims to take both location and text information into account during tree construction, by optimizing for a combination of minimizing the areas of the enclosing rectangles and maximizing the text similarities between the documents of the enclosing rectangles.

We present an algorithm for building the DIR-tree in Sect. 5.1.1, and we cover query processing in Sect. 5.1.2.

5.1.1 The DIR-tree

The structure of nodes in a DIR-tree is similar to that of nodes in an IR-tree. The only difference is that each non-leaf entry e additionally has a centroid document $e.\phi$ of all the documents enclosed in the subtree rooted at e.

Formally, let $\phi = \langle w_1, \ldots, w_m \rangle$ be the centroid document of a set of documents $\{\psi_1, \ldots, \psi_n\}$. Then, $\phi.w_i = \max(\psi_1.w_i, \ldots, \psi_n.w_i)$. To choose an appropriate insertion path for an object, the DIR-tree takes into account both the spatial area parameter and the similarity between the document of the object and the centroid document of the entry in a node. We next describe how to incorporate the document similarity.

Let $e_1, \ldots, e_p$ be the entries in the current node, and let O be the object to be inserted. In the R-tree, the spatial area cost of inserting O into e_k, $1 \leq k \leq p$, is defined as $AreaCost(e_k) = area(e'_k.r) - area(e_k.r)$, where $e'_k.r$ is the (possibly enlarged) version of rectangle e_k after inclusion of O.

Definition 2 The spatial area cost extended with document similarity is defined as follows:

$$SimAreaCost(e_k, O) = (1-\beta)\frac{AreaCost(e_k)}{maxArea} + \beta(1 - DocSim(e_k.\phi, O.\psi)) \quad (4)$$

In this definition, the document similarity $DocSim(\cdot,\cdot)$ can be any measure (e.g., cosine similarity and Hamming distance) of the distance between document vectors,[1] $maxArea$, the area of the minimum bounding rectangle enclosing all objects, is used for normalization, and β is a parameter. Observe that if we set $\beta = 0$, the DIR-tree reduces to the IR-tree. In the other extreme, setting $\beta = 1$ means that we consider only document similarity when building a DIR-tree.

The insertion algorithm of the DIR-tree follows that of the IR-tree, with the exception of the specifics of functions ChooseSubtree and Split. Function ChooseSubtree is given in Algorithm 3. Beginning at the root, the function finds at every level the most suitable subtree to accommodate the new entry until a leaf node is reached. Specifically, it repeatedly chooses subtrees that minimize Eq. 4, that is, the value of $SimAreaCost(\cdot,\cdot)$. If ChooseSubtree reaches a leaf node with the maximum number of entries M, function Split distributes the $M+1$ rectangles between two nodes. We incorporate document similarity into the standard Quadratic Split algorithm [16]. Function Split is given in Algorithm 4.

5.1.2 Query processing in the DIR-tree

We use Algorithm 1 for the processing of LkT queries on the DIR-tree. An example illustrates its benefits.

Example 6 We build a DIR-tree on the 8 objects in Fig. 1. The result is shown in Figs. 3 and 4. Consider the query Q in Fig. 3, where $Q.\psi =$ (Chinese restaurant). If we apply Algorithm 1 to retrieve the top-2 most relevant objects, the result is $\langle O_1, O_5\rangle$, and the nodes *root*, R_5, and R_1 are visited in the DIR-tree. If we use instead the corresponding IR-tree in Fig. 2, nodes *root*, R_5, R_6, R_1, and R_3 are visited. The DIR-tree thus yields better performance than does the IR-tree for query Q.

[1] Unlike in query processing, we do not use language models here since this would introduce asymmetry.

Algorithm 3 ChooseSubtree

```
1: N ← the root;
2: loop
3:     if N is a leaf node then
4:         Return N;
5:     else
6:         Choose the entry in N with the minimum value for
       SimAreaCost(e_k, O), resolving ties by choosing the entry with the
       minimum value for AreaCost(e_k);
7:         N ← the child node pointed by the child pointer of the chosen
       entry;
```

Algorithm 4 Split(N)

```
1: for each pair of entries e_i and e_j in node N do
2:     R_ij ← compose a rectangle including e_i and e_j;
3:     d_ij ← area(R_ij) − area(e_i) − area(e_j);
4:     DocSim_ij ← similarity between documents of e_i and e_j;
5:     ineffi ← (1 − β)d + β(1 − DocSim);
6: Choose the pair with the largest ineffi value to be the first elements
   of the two groups;
7: loop                                  ▷ assign all entries in N to the two groups
8:     if all entries in N have been assigned to the two groups then
9:         break;
10:    if one group needs to include all remaining entries then
11:        Assign all remaining entries to it;
12:        break;
13:    Choose the next entry and add it to the group with the smaller
       SimAreaCost(·,·), resolving ties by adding the entry to the group
       with the smaller AreaCost();
```

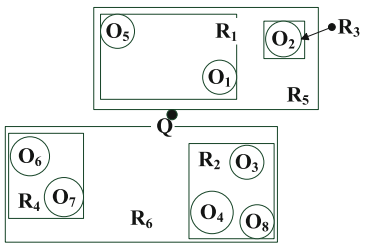

Fig. 3 Objects and bounding rectangles

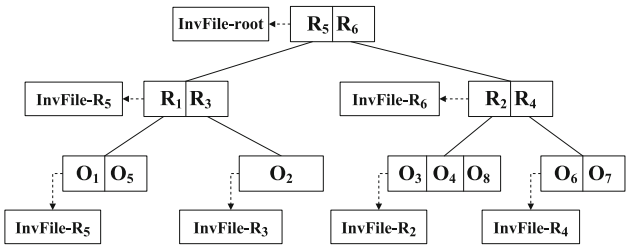

Fig. 4 Example of the DIR-tree

5.2 Cluster-enhanced method

We propose to enhance the hybrid indexing framework with clustering, which makes it possible to estimate tighter bounds at each tree node, thus improving query performance.

Spatial web objects often belong to different categories. For example, the geo-referenced points of interest may

belong to specific categories, such as retail, accommodations, restaurants, and tourist attractions. Points of interest from different such categories may appear in the same node of the hybrid index, and thus two objects in the same node can contain very dissimilar document.

The idea is to cluster objects into groups according to their documents. Each index node may then contain objects from different clusters. Instead of constructing a single pseudo document for each node, we construct a pseudo document for each cluster in each node. Since objects within the same group are more similar than objects in different groups, the bounds estimated using clusters in a node will be tighter than those estimated for whole nodes. Therefore, we may expect the use of clustering to improve the query performance of both the IR-tree and the DIR-tree. We name the cluster-enhanced IR-tree the CIR-tree and the cluster-enhanced DIR-tree the CDIR-tree.

5.2.1 The CIR-tree

For convenience of presentation, we describe the proposed cluster-enhanced method in the context of the IR-tree; the method is equally applicable to the DIR-tree. Given any clustering of objects into n groups $C_1, \ldots, C_n$, we form an inverted file incorporating the cluster information. We next present how to incorporate the cluster information into the nodes in the hybrid index.

The structure of a leaf node in the CIR-tree is similar to that of leaf nodes in the IR-tree. The only difference is that we add a cluster label to each entry (object) in the leaf node. The cluster label indicates the cluster the object belongs to.

Next, for each non-leaf entry e, we construct a set of pseudo documents. Each such document corresponds to one cluster. The number of pseudo documents of e depends on the number of clusters that objects in the subtree of e belong to. The inverted file of a non-leaf node indexes all these pseudo documents.

The pseudo document of a cluster at a node is constructed in a bottom-up manner similarly to how we generate the pseudo documents in the IR-tree. Specifically, for each cluster C_i in a leaf node pointed to by a non-leaf entry e, we first construct a pseudo document denoted by $e.C_i.\psi$. For each term t, we choose the maximum weight of t in all the documents in each cluster C_i in the leaf node as the weight of t in $e.C_i.\psi$.

Having constructed the pseudo documents of the clusters of the leaf nodes, we can construct pseudo documents of clusters of nodes at the upper levels from bottom to top. When we use pseudo documents at a lower level to construct pseudo documents at a higher level, identical clusters from different child nodes should be combined.

For instance, assume that a non-leaf entry e has two child entries e_a and e_b (i.e., child nodes), where e_a includes two clusters $\{e_a.C_1, e_a.C_2\}$, and where $\{e_b.C_2, e_b.C_3\}$ are the two clusters in e_b. The entry e should include three, not four, clusters, namely $\{e.C_1, e.C_2, e.C_3\}$. We combine $e_a.C_2$ and $e_b.C_2$ to obtain $e.C_2$. For each term t, its weight is the maximum value of the weight of t in $e_a.C_2.\psi$ and the weight of t in $e_b.C_2.\psi$.

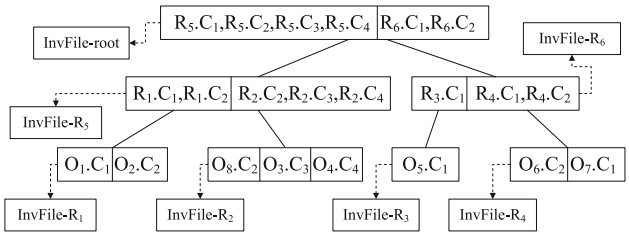

Fig. 5 Example of the CIR-tree

Example 7 Consider again the 8 objects in Fig. 1. We cluster these documents into four clusters: $C_1 = \{O_1, O_5, O_7\}$, $C_2 = \{O_2, O_6, O_8\}, C_3 = \{O_3\}, C_4 = \{O_4\}$. Figure 5 shows the CIR-tree for the 8 objects. The contents of InvFile-R_1, InvFile-R_2, InvFile-R_3, and InvFile-R_4 are the same as in Table 5. The contents of InvFile-R_5, InvFile-R_6, and InvFile-root are given in Table 7.

5.2.2 Query processing in the CIR-tree

The algorithm for processing LkT queries on the CIR-tree is almost the same as Algorithm 2, the exception being that line 18 in Algorithm 2 is replaced by the code in Fig. 6. The key of an entry in the priority queue is the minimum value of $MIND_{ST}(Query, Node.C_i)$ among all clusters.

Example 8 Consider the query in Example 5. The algorithm for processing the query using the CIR-tree starts by enqueueing the entries in the root and then executes the following steps:

(1) Queue:{(R_5,0.4),(R_6,0.495)}
Dequeue R_5, enqueue R_1 and R_2.
(2) Queue:{(R_1,0.475),(R_6,0.495),(R_2,0.59965)}
Dequeue R_1, enqueue O_1 and O_2.
(3) Queue:{(O_1,0.475),(R_6,0.495),(R_2,0.59965),
(O_2,0.74975)}
Dequeue O_1. Report O_1 as the top-1 object. Terminate.

Observe that before reporting O_1, nodes *root*, R_5, and R_1 are visited. Compared with the algorithm using the IR-tree, fewer nodes are visited (recall that the algorithm using the IR-tree also visited R_2) because the CIR-tree provides tight bounds.

5.2.3 Clustering objects

Ideally, we would find clusters that are optimal for the running time of Algorithm 2. Consider a query Q for the top-k objects on dataset $\mathcal{D}$ and a set of clusters $\{C_1, \ldots, C_n\}$ on the

Table 7 Contents of inverted files in the CIR-tree

	Chinese	Spanish	Restaurant	Food
InvFile-R_5	$\langle R_1.C_1, 0.5\rangle$	$\langle R_1.C_2, 0.5\rangle$	$\langle R_1.C_1, 0.5\rangle$	$\langle R_2.C_3, 0.1\rangle$
	$\langle R_2.C_3, 0.7\rangle$	$\langle R_2.C_2, 0.3\rangle$	$\langle R_1.C_2, 0.5\rangle$	$\langle R_2.C_4, 0.1\rangle$
			$\langle R_2.C_2, 0.3\rangle$	
			$\langle R_2.C_4, 0.7\rangle$	
InvFile-R_6	$\langle R_3.C_1, 0.4\rangle$	$\langle R_4.C_1, 0.1\rangle$	$\langle R_3.C_1, 0.4\rangle$	$\langle R_4.C_1, 0.1\rangle$
	$\langle R_4.C_1, 0.1\rangle$	$\langle R_4.C_2, 0.4\rangle$	$\langle R_4.C_1, 0.4\rangle$	
			$\langle R_4.C_2, 0.3\rangle$	
InvFile-root	$\langle R_5.C_1, 0.5\rangle$	$\langle R_5.C_2, 0.5\rangle$	$\langle R_5.C_1, 0.5\rangle$	$\langle R_5.C_3, 0.1\rangle$
	$\langle R_5.C_3, 0.7\rangle$	$\langle R_6.C_1, 0.1\rangle$	$\langle R_5.C_2, 0.5\rangle$	$\langle R_5.C_4, 0.1\rangle$
	$\langle R_6.C_1, 0.4\rangle$	$\langle R_6.C_2, 0.4\rangle$	$\langle R_5.C_4, 0.7\rangle$	$\langle R_6.C_1, 0.1\rangle$
			$\langle R_6.C_1, 0.4\rangle$	
			$\langle R_6.C_2, 0.3\rangle$	

$minD = \min_{(1 \le i \le m)} MIND_{ST}(Query, Node.C_i)$;
$Queue$.Enqueue($Node, minD$);
▷ m is the number of clusters in $Node$

Fig. 6 Replacement to line 18 in Algorithm 2

documents in $\mathscr{D}$. Let ScanTime($C_1, \ldots, C_n, \mathscr{D}, k, Q$) be the number of objects checked by Algorithm 2 before returning the top-k objects. Given ($\mathscr{D}$, k, Q), one would like to find n clusters $C_1, \ldots, C_n$ such that ScanTime($C_1, \ldots, C_n, \mathscr{D}, k$, Q) is minimized.

It is well known that finding clusters of points that minimize the diameter within a metric space is NP-hard [15]. Although minimizing the diameter of a cluster in our problem usually results in a tight upper bound of each cluster for a query, this does not immediately imply that finding a clustering solution that minimizes ScanTime() is NP-hard, since ScanTime() does not directly correspond to the diameters of clusters.

Theorem 2 *The problem of finding a clustering solution that minimizes* ScanTime*($C_1, \ldots, C_n, \mathscr{D}, k, Q$) is NP-hard.*

Proof The proof is by reduction from the *bin packing problem*, which is NP-hard [15]. Consider $k = 1$ and let objects have the same location information. If the *IRScore* computed based on pseudo documents (i.e., upper bounds) is smaller than the *IRScore* of the top-1 document, Algorithm 2 first scans the clusters containing the top-1 document to find the top-1 document, and it then prunes the clusters whose upper bound *IRScore* is lower than the *IRScore* of the top-1 document.

Assume that a parameter B is the sum of the logarithmic value of the term language model of the top-1 result. Minimizing the scan time corresponds to the decision problem of assigning all objects to k clusters, such that for each cluster, $Score(C) \le B$. Given an instance of the bin packing problem, each item corresponds to one object whose document contains a distinct term, each bin corresponds to a cluster, and the size of each item corresponds to the logarithmic value of the term language model. The query Q will contain all terms in documents in $\mathscr{D}$. There is then a solution for the bin packing problem that packs all the items in k bins of size B if and only if there is a solution for our problem. □

Given the above result, we must use a heuristic method for the clustering. One natural and simple approach is to use the k-means clustering algorithm [23].

5.3 ClusterMBR enhancement

The cluster-enhanced method proposed in Sect. 5.2 provides tighter bounds on document similarities in each node. However, each cluster in the same node has the same MBR, which is the MBR of the node itself. We propose to refine the MBR of each cluster by constructing an MBR for each cluster at a node. This enables us to estimate a tighter bound on the distance from the query to the node. This technique can be applied to both the CIR-tree and the CDIR-tree.

For convenience of presentation, we introduce the ClusterMBR technique in the context of the CIR-tree. We construct cluster MBRs only in non-leaf nodes of the CIR-tree while the structure of leaf nodes remains the same. The MBR of a cluster in a non-leaf node is constructed in a bottom-up manner as we generate the MBRs in the R-tree. Specifically, for each non-leaf entry e that includes n clusters, we construct n MBRs, one for each cluster. The MBR of cluster C_i encloses the rectangles of the child entries that belong to the same cluster C_i. In order not to affect the tree structure of the CIR-tree, the cluster MBRs for each non-leaf node are stored

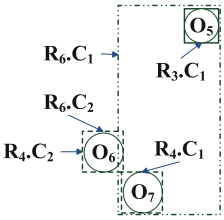

Fig. 7 Example of cluster MBRs

separately and are connected to the corresponding non-leaf node by a pointer.

Example 9 Consider the right branch of the CIR-tree shown in Fig. 5. Figure 7 shows the cluster MBRs of non-leaf entries R_3, R_4, and R_6. As an example, the MBR of cluster C_1 for entry R_6 encloses the MBR of $R_3.C_1$ and $R_4.C_1$. Comparing Figs. 1 and 7, the cluster MBRs are smaller than the original MBRs, and they thus provide tighter bounds.

When applying the ClusterMBR technique on the CIR-tree, the algorithm for processing LkT queries is almost the same as the algorithm used on the CIR-tree. The only difference is that when computing $MIND_{ST}(Query, Node.C_i)$, we use the MBR of C_i instead of the MBR of the entire *Node*.

5.4 TermMBR enhancement

We proceed to propose the TermMBR Enhancement to optimize our framework. Each term in an inverted file has multiple locations in an MBR of a node, that is, a term locates at the locations of objects that have documents that include the term. Consider R_5 in Fig. 1: documents $O_1.\psi$ and $O_3.\psi$ include the term Chinese, so in rectangle R_5, term Chinese has the two locations of O_1 and O_3. In our framework, given a query Q, when computing $MIND_{ST}$ of one entry e in a node, the minimum distance between $Q.\lambda$ and $e.r$, $mindist(Q.\lambda, e.r)$, serves as the lower bound on distance between Q and all objects in the rectangle $e.r$. However, this lower bound is loose because each term in $Q.\lambda$ has its own locations. We only need to consider these locations, and other places in the rectangle do not contribute to the query. Hence, for each non-leaf entry in the index structure, we draw an MBR for each term that encloses all the locations of that term. Figure 8 shows the MBRs of the four terms in R_5 as given in Fig. 1.

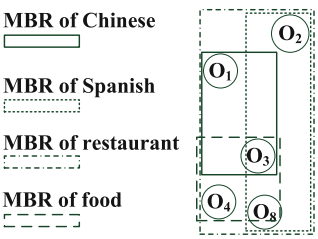

Fig. 8 Example of term MBRs

Given a query Q, when computing $MIND_{ST}$ of a non-leaf entry e, we fetch the MBRs of all terms in $Q.\psi$. The minimum distance between $Q.\lambda$ and those MBRs is then used to replace $mindist(Q.\lambda, e.r)$. By doing so, the lower bound is expected to be tighter and to lead to better query performance. The term MBRs are added in the posting lists of the inverted files.

5.5 Update of the IR-tree

Since the proposed index structures are based on the R-tree, the update in the framework, including insertion and deletion of an object, is similar to the corresponding operations in the R-tree. For the IR-tree based indexes, insertion and deletion of an object in the tree structure is exactly as in the R-tree. In addition, corresponding information in the inverted files, cluster MBRs, or term MBRs can be updated easily. For DIR-tree based indexes, Algorithm 3 and 4 are used for inserting or deleting an object. Additional information related to an inserted or deleted object can be updated as in the IR-tree based indexes.

6 Processing of RkT queries

The RkT^e query can be processed by the proposed algorithms for the LkT query. We thus proceed to focus on the RkT^u query. Section 6.1 proposes an in-memory algorithm for the processing of the RkT^u query. Section 6.2 proposes to use a disk resident index, that is, IR-tree, to find candidates first, And then those candidates are passed to the algorithm described in Sect. 6.1 to compute the final result.

6.1 The RkT^u Algorithm

Given an RkT^u query Q, since $Q.\Omega$ is a spatial extent, the whole dataset $\mathscr{D}$ is partitioned into two sets. One set $\mathscr{D}_{in}$ contains objects that locate inside $Q.\Omega$. The other set $\mathscr{D}_{out}$ contains objects that locate outside $Q.\Omega$. The result of Q may contain some objects that belong to $\mathscr{D}_{in}$ (*internal top-k objects*) and some objects that belong to $\mathscr{D}_{out}$ (*external top-k objects*).

An RkT^u query Q can be represented by a set of LkT queries $\mathscr{Q} = \{Q_1, Q_2, \ldots\}$, where the union of the spatial components of all the LkT queries in $\mathscr{Q}$ equals to the spatial component of the RkT^u query, that is, $\bigcup_{Q_i \in \mathscr{Q}} \{Q_i.\lambda\} = Q.\Omega$ and all the LkT queries in $\mathscr{Q}$ have the same keywords as does the RkT^u query, that is, $\forall Q_i \in \mathscr{Q}(Q_i.\psi = Q.\psi)$. We first consider the case $k = 1$. We propose two lemmas that indicate the necessary and sufficient conditions for an object to be an internal or an external top-1 object.

Lemma 1, below, is exploited when computing internal results of RkT^u queries.

Lemma 1 *An object O is an internal top-1 object of the RkT^u query Q if and only if O is located inside the query region $Q.\Omega$ and it is the top-1 object of an LkT query $Q_i(\in \mathcal{Q})$, where Q_i has the same location as does the object O. Formally, object O is an internal top-1 object of the RkT^u query $Q \iff O.\lambda \in Q.\Omega$ and O is the top-1 object of one LkT query $Q_i(\in \mathcal{Q})$ where $Q_i.\lambda = O.\lambda$ and $Q_i.\psi = Q.\psi$.*

Proof The proof of the implication $\Longleftarrow$ is inherent in the definition of the RkT^u query. The proof of the implication $\Longrightarrow$ is done by contradiction. Assume that an object O' other than O is the top-1 object of Q_i, as shown in Fig. 9a. According to the definition of the LkT query, we have:

$$\alpha\frac{D_\varepsilon(Q_i.\lambda, O.\lambda)}{maxD} + (1-\alpha)\left(1 - \frac{P(Q_i.\psi|O.\psi)}{maxP}\right) > \alpha\frac{D_\varepsilon(Q_i.\lambda, O'.\lambda)}{maxD} + (1-\alpha)\left(1 - \frac{P(Q_i.\psi|O'.\psi)}{maxP}\right). \quad (5)$$

Since $Q_i.\lambda = O.\lambda$, that is, $D_\varepsilon(Q_i.\lambda, O.\lambda) = 0$, Eq. 5 becomes:

$$(1-\alpha)\left(1 - \frac{P(Q_i.\psi|O.\psi)}{maxP}\right) > \alpha\frac{D_\varepsilon(Q_i.\lambda, O'.\lambda)}{maxD} + (1-\alpha)\left(1 - \frac{P(Q_i.\psi|O'.\psi)}{maxP}\right). \quad (6)$$

Since O is an internal top-1 object of Q, it must be the top-1 object of another LkT query $Q_j(\in \mathcal{Q})$. Then we have:

$$D_{ST}(Q_j, O) = \alpha\frac{D_\varepsilon(Q_j.\lambda, O.\lambda)}{maxD} + (1-\alpha)\left(1 - \frac{P(Q_j.\psi|O.\psi)}{maxP}\right). \quad (7)$$

Since $Q_i.\psi = Q.\psi$ and $Q_j.\psi = Q.\psi$, combining Eq. 6 and 7, we have:

$$D_{ST}(Q_j, O) > \alpha\frac{D_\varepsilon(Q_j.\lambda, O.\lambda) + D_\varepsilon(Q_i.\lambda, O'.\lambda)}{maxD} + (1-\alpha)\left(1 - \frac{P(Q_j.\psi|O'.\psi)}{maxP}\right) \quad (8)$$

Since $Q_i.\lambda = O.\lambda$ and according to the triangle inequality, that is, $D_\varepsilon(Q_j.\lambda, O.\lambda) + D_\varepsilon(O.\lambda, O'.\lambda) > D_\varepsilon(Q_j.\lambda, O'.\lambda)$, Eq. 8 becomes:

$$\begin{aligned} D_{ST}(Q_j, O) &> \alpha\frac{D_\varepsilon(Q_j.\lambda, O'.\lambda)}{maxD} + (1-\alpha)\left(1 - \frac{P(Q_j.\psi|O'.\psi)}{maxP}\right) \\ &= D_{ST}(Q_j, O'). \end{aligned}$$

Hence, object O is not the top-1 object of the LkT query Q_j. This contradicts the assumption that object O is the top-1 object of the LkT query Q_j and completes the proof. □

Lemma 2, below, is used when computing external results of RkT^u queries.

Lemma 2 *An object O is an external top-1 object of the RkT^u query Q if it is located outside Q's region and is the top-1 object of at least one LkT query that has the same keywords and is located on the boundary of Q's region. Formally, object O is an external top-1 object of the RkT^u query $Q \iff O.\lambda \notin Q.\Omega$ and O is the top-1 object of at least one LkT query $Q_i(\in \mathcal{Q})$ such that $Q_i.\lambda$ belongs to the boundary of $Q.\Omega$ and $Q_i.\psi = Q.\psi$.*

Proof The proof of the implication $\Longleftarrow$ is inherent in the definition of the RkT^u query. We prove the implication $\Longrightarrow$ by contradiction. Assume object O is not the top-1 object of any LkT query in $\mathcal{Q}$ that locates on the boundary of $Q.\Omega$. Since object O is an external top-1 object of the RkT^u query Q, object O must be the top-1 object of at least one LkT query $Q_j(\in \mathcal{Q})$ inside $Q.\Omega$. As shown in Fig. 9b, let Q_i be the LkT query in $\mathcal{Q}$ that locates at the intersection of segment $\overline{O.\lambda Q_j.\lambda}$ and the boundary of $Q.\Omega$. Since O is not the top-1 object of the LkT query Q_i, there must be an object O' that is the top-1 object of Q_i, that is,

$$\alpha\frac{D_\varepsilon(Q_i.\lambda, O.\lambda)}{maxD} + (1-\alpha)\left(1 - \frac{P(Q_i.\psi|O.\psi)}{maxP}\right) > \alpha\frac{D_\varepsilon(Q_i.\lambda, O'.\lambda)}{maxD} + (1-\alpha)\left(1 - \frac{P(Q_i.\psi|O'.\psi)}{maxP}\right). \quad (9)$$

Adding $\alpha\frac{D_\varepsilon(Q_i.\lambda, Q_j.\lambda)}{maxD}$ on both sides of Eq. 9, since

$$D_\varepsilon(Q_i.\lambda, O'.\lambda) + D_\varepsilon(Q_i.\lambda, Q_j.\lambda) > D_\varepsilon(Q_j.\lambda, O'.\lambda),$$

and $Q_i.\psi = Q_j.\psi$, we have

$$\alpha\frac{D_\varepsilon(Q_j.\lambda, O.\lambda)}{maxD} + (1-\alpha)\left(1 - \frac{P(Q_j.\psi|O.\psi)}{maxP}\right) > \alpha\frac{D_\varepsilon(Q_j.\lambda, O'.\lambda)}{maxD} + (1-\alpha)\left(1 - \frac{P(Q_j.\psi|O'.\psi)}{maxP}\right),$$

that is, $D_{ST}(Q_j, O) > D_{ST}(Q_j, O')$. Hence, object O is not the top-1 object of Q_j, which contradicts the assumption that O is the top-1 object of Q_j. □

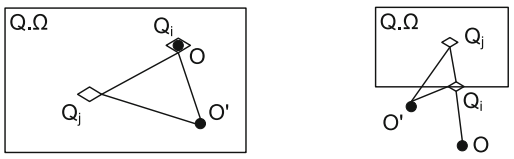

Fig. 9 Proof of Lemmas 1 and 2

Lemma 2 shows that finding external top-1 objects of an RkT^u query Q is equivalent to finding the top-1 object for LkT queries in $\mathscr{Q}$ that locate on the boundary of $Q.\Omega$. We assume the boundary of $Q.\Omega$ comprises four line segments. The problem is converted into finding top-1 objects for an RkT^u query Q where $Q.\Omega$ degenerates to four line segments, where each line segment $Q.L$ is represented by two pairs of coordinates: $[Q.L.minx, Q.L.miny : Q.L.maxx, Q.L.maxy]$.

Before describing how to find external top-1 objects for LkT queries in $\mathscr{Q}$ that locate on the boundary of $Q.\Omega$, we first study the ranking function of the RkT^u query. Contrary to the LkT query, an object O does not have only one constant ranking score, but has multiple ranking scores described by a function $f(q, O)$ for a given RkT^u query Q. Lemma 3 explains this function and its properties.

Lemma 3 *Given an RkT^u query Q, each object O has a ranking score function $f(q, O)$ for each line segment $Q.L$ of the boundary of $Q.\Omega$ that models its ranking scores, where q is a point location on $Q.L$.*

Proof According to Eq. 2, given an RkT^u query Q, for each object O, the value of function $P(Q.\psi|O.\psi)$ is a constant, while $D_{ST}(Q, O)$ is a function of q ($q \in Q.L$), that is,

$$f(q, O) = \alpha \frac{\sqrt{(q.x - O.x)^2 + (q.y - O.y)^2}}{maxD} + T(O.\psi), \tag{10}$$

where $T(O.\psi) = (1-\alpha)(1 - \frac{P(Q.\psi|O.\psi)}{maxP})$. Since the line segment $Q.L$ is either horizontal or vertical, either $q.y$ or $q.x$ is fixed. Without loss of generality, we assume $Q.L$ is horizontal and $q.y$ is a constant $L.y$. Then Eq. 10 becomes:

$$f(q, O) = B\sqrt{(q.x - O.x)^2 + A(O.y)} + T(O.\psi), \tag{11}$$

where $A(O.y) = (L.y - O.y)^2$ and $B = \frac{\alpha}{maxD}$.

The ranking score function $f(q, O)$ is one arm of a hyperbola. Taking the first derivative and setting it equal to 0, we get:

$$B\frac{(q.x - O.x)}{\sqrt{(q.x - O.x)^2 + A(O.y)}} = 0.$$

Solving for $q.x$ gives $q.x = O.x$. The second derivative is

$$B\frac{A(O.y)}{\sqrt{((q.x - O.x)^2 + A(O.y))^3}}.$$

Since it is positive, $q.x = O.x$ is a minimum. Hence, function $f(q, O)$ is monotone decreasing on $(-\infty, O.x]$ and monotone increasing on $[O.x, +\infty]$. □

Figure 10 shows the curves of ranking score functions of four objects with respect to a line segment $[0.2, 0.5 : 0.3, 0.5]$.

Given an RkT^u query Q, for each line segmant $Q.L$ of Q, let $\mathscr{Q} = \{Q_1, Q_2, \ldots\}$ be the set of LkT queries such that

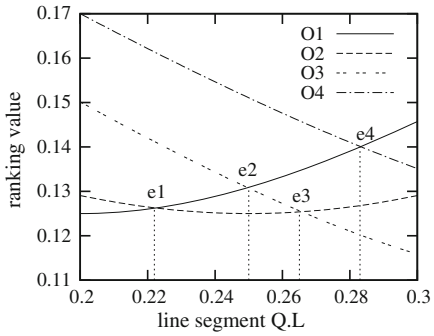

Fig. 10 $D_{ST}(Q, O)$ as a function of points on $Q.L$

the union of the spatial components of all the LkT queries in $\mathscr{Q}$ equals to the line segment $Q.L$, that is, $\bigcup_{Q_i \in \mathscr{Q}}\{Q_i.\lambda\} = Q.L$ and all the LkT queries in $\mathscr{Q}$ have the same keywords as does the RkT^u query, that is, $\forall Q_i \in \mathscr{Q}(Q_i.\psi = Q.\psi)$. For each object O, we derive its ranking score function by following Eq. 11. Let $E = \{e_1, e_2, \ldots\}$ denote the set of intersection points of the ranking functions of any pair of objects in $\mathscr{D}$. We arrange the intersections in E in the order of their projected values on $Q.L$, and we then divide the line segment $Q.L$ into a finite number of subsegments $Q.L = \{L_1, L_2, \ldots\}$ according to E. In Fig. 10, four intersections $e1, \ldots, e4$ divide $Q.L$ into five subsegments. We next present Lemmas 4 and 5, showing that the top-1 objects of LkT queries, which belong $\mathscr{Q}$ and locate on the same subsegment, are the same, and the union of the results of all subsegments are the result of LkT queries on $Q.L$.

Lemma 4 *The LkT queries in $\mathscr{Q}$ that are located on the same subsegment have the same top-1 object, that is,*

$$\forall L_m(Q_i.\lambda \in L_m \wedge Q_j.\lambda \in L_m \wedge Q_i \in \mathscr{Q} \wedge Q_j \in \mathscr{Q} \implies LkT(Q_i.\lambda, Q_i.\psi) = LkT(Q_j.\lambda, Q_j.\psi)) \tag{12}$$

Proof Given any two objects O_1 and O_2 in $\mathscr{D}$, in order to find the intersections between their ranking functions, equation $f(q.x, O_1) = f(q.x, O_2)$ needs to be solved, that is,

$$B\sqrt{(q.x - O_1.x)^2 + A(O_1.y)} + T(O_1.\psi) = B\sqrt{(q.x - O_2.x)^2 + A(O_2.y)} + T(O_2.\psi) \tag{13}$$

Let $(q.x - O_1.x)^2 + A(O_1.y) = z^2$, where $z > 0$. Equation 13 becomes:

$$4(O_1.x - O_2.x)^2 z^2 - 2Bz + C = 0, \tag{14}$$

where $C = (O_1.x - O_2.x)^2(B^2 - 4A(O_1.y)) + B^2(A(O_1.y) - A(O_2.y)) - (T(O_1.\psi) - T(O_2.\psi))^2$.

Since Eq. 14 has two roots at most, there are at most four different values of $q.x$, that is, there are at most four intersections between any two ranking functions. Since $\mathscr{D}$ is a finite set, intersection set E is also finite. Hence, line segment $Q.L$ is divided into a finite number of subsegments by E.

We prove Eq. 12 by contradiction. Given any subsegment L_m, suppose LkT queries Q_1 and Q_2 locate on L_m and have object O_1 and object O_2 as their top-1 object, respectively. Without loss of generality, assume $Q_1.x < Q_2.x$. Then we have $f(Q_1.x, O_1) < f(Q_1.x, O_2)$ and $f(Q_2.x, O_2) < f(Q_2.x, O_1)$. Since function $f(\cdot, \cdot)$ is continuous, there must be an intersection between $f(q.x, O_1)$ and $f(q.x, O_2)$ when $q.x \in (Q_1.x, Q_2.x)$. That contradicts to the definition of subsegment L_m. □

As an example in Fig. 10, subsegment $[Q.L.minx, e1.x]$ takes $O1$ as the top-1 object. The top-1 object of subsegment $[e1.x, e2.x]$ is $O2$.

Definition 3 An intersection e_i ($e_i \in E$) is called a change point if the top-1 object of LkT queries located on subsegment $[e_{i-1}.x, e_i.x)$ is different from the top-1 object of LkT queries located on subsegment $(e_i.x, e_{i+1}.x]$.

Lemma 5 *Let O be the top-1 object of LkT queries located on subsegment $[e_{i-1}.x, e_i.x)$. If e_i is an intersection between the ranking functions of two different objects O and O' then e_i is a change point, and object O' is the top-1 object of LkT queries located on subsegment $(e_i.x, e_{i+1}.x]$.*

Proof Let $f(q.x, O)$ be the ranking function of O and let $f(q.x, O')$ be the ranking function of O'. Consider function $g(x) = f(q.x, O) - f(q.x, O')$. Since e_i is the intersection between $f(q.x, O)$ and $f(q.x, O')$, we have $g(e_i.x) = 0$. For a small value h, $g(e_i.x - h) = f(e_i.x - h, O) - f(e_i.x - h, O') < 0$, since O is the top-1 object of subsegment $[e_{i-1}.x, e_i.x)$. Thus, we have $g'(e_i.x) > 0$. Therefore, we know $g(e_i.x + h) > 0$, that is, $f(e_i.x + h, O) > f(e_i.x + h, O')$ because $g(x)$ is a continuous function. Hence, object O is not the top-1 object of subsegment $(e_i.x, e_{i+1}.x]$, and e_i is a change point.

Let O'' be the top-1 object of subsegment $(e_i.x, e_{i+1}.x]$ instead of O', and let $f(q.x, O'')$ be the ranking function of O''. Since $f(x, O) < f(x, O'')$ when $x \in [e_{i-1}.x, e_i.x)$, $f(x, O) > f(x, O'')$ when $x \in (e_i.x, e_{i+1}.x]$, and both $f(x, O)$ and $f(x, O'')$ are continuous, there must be at least one intersection between $f(x, O)$ and $f(x, O'')$ on $[e_{i-1}.x, e_{i+1}.x]$. That contradicts the premise that e_i is the only intersection on subsegment $[e_{i-1}.x, e_{i+1}.x]$. Hence, object O' is the top-1 object of subsegment $(e_i.x, e_{i+1}.x]$. □

Based on Lemmas 2, 4, and 5, we proceed to describe how to find the top-1 objects for RkT^u queries. Starting from the left-hand side endpoint of $Q.L$, for each object O_i in $\mathscr{D}$, we compute its value of $f(Q.L.minx, O_i)$. The object that has the smallest value is added to the result and taken as the current top-1 object O. Next, we find the subsegment on which the LkT queries have the same top-1 object O. We compute the intersections between the ranking function of O and all the other objects. The intersection e that falls into the range $[Q.L.minx, Q.L.maxx]$ and has the smallest x value is the change point, and thus LkT queries located on subsegment $[Q.L.minx, e.x]$ take O as the top-1 object. The corresponding object O' of e (the ranking functions of O and O' intersect at e) is taken as the current top-1 object. Then, the same process is repeated to find the next subsegment starting from $e.x$ on which the LkT queries have the same current top-1 object. The algorithm stops when no new change point is found. The set of all found top-1 objects are returned as the result.

Example 10 As shown in Fig. 10, $Q.L.minx = 0.2$, and $f(Q.L.minx, O_1)$ is the smallest. Object O_1 is the current top-1 object. Since intersection $e1$ (between the ranking functions of O_1 and O_2) is a change point, object O_1 is the top-1 object of subsegment $[Q.L.minx, e1.x)$. Next, we find the subsegment $(e1.x, e3.x)$ that takes O_2 as the top-1 object, since intersection $e3$ (between the ranking functions of O_2 and O_3) is a change point. Finally, object O_3 is the top-1 object of subsegment $(e3.x, Q.L.maxx]$.

Extension to k Lemmas 2, 4, and 5 hold when $k > 1$. Algorithm 5 shows the pseudocode for retrieving external top-k objects of one of the line segments of the boundary of an RkT^u query Q. The union of the results of four line segments is the final result of the RkT^u query Q. Assume the line segment is horizontal. It is easy to apply the algorithm to vertical line segments. Line 1 computes the ranking score of each object by using the Euclidean distance between the leftmost point $Q.L.minx$ of the line segment and objects. The k objects, denoted by $TOPK$, with the smallest ranking score are added to the result set. The change point cp is set to $Q.L.minx$ (Line 2). Next, it computes the intersections between the ranking function of each object in $TOPK$ and the ranking function of each object in $\mathscr{D} \setminus TOPK$ (Lines 4–6). The intersections that fall between the change point and the rightmost point of the line segment are kept in E (Lines 7–9). If E is empty, the top-k objects of the line segment have been found, and the algorithm stops. Otherwise, we select the leftmost intersection e in E. Suppose e is the intersection between ranking functions of object $O (\in TOPK)$ and $O' (\in \mathscr{D} \setminus TOPK)$. Object O is removed from the current top-k set $TOPK$. Object O' is added to $TOPK$ and the result set. The change point is updated to $e.x$ (Lines 12–16).

6.2 Using the IR-tree for pruning

Using the RkT^u algorithm, it is computational expensive to find change points when the dataset is large, since it needs to compute the intersections between the ranking function of the current top-k object and that of all the other objects. We propose to use the IR-tree to find candidates that are a small

Algorithm 5 RkT$^u(Q, k)$

```
1: Select k objects (denoted by TOPK) and add them to RESULT, such that ∀O_i ∈ 𝒟 \ TOPK ∀O_j ∈ TOPK(f(Q.L.minx, O_j) < f(Q.L.minx, O_i))
2: cp ← Q.L.minx;
3: while true do
4:    for each object O_i in 𝒟 \ TOPK do
5:       for each object O_j in TOPK do
6:          Compute intersections between f(q.x, O_j) and f(q.x, O_i);
7:          for each intersection e do
8:             if cp ≤ e.x ≤ Q.L.maxx then
9:                Add e to E;
10:   if E is empty then
11:      break;
12:   Select the intersection e in E that has the smallest x coordinate;
      ▷ e is the intersection between ranking functions of O and O'
13:   Remove O from TOPK;
14:   Add O' to TOPK;
15:   Add O' to RESULT;
16:   cp ← e.x;
17: return RESULT;
```

portion of the whole dataset. Then, the RkT^u algorithm is applied on the small set of candidates to find the true results.

According to Eq. 11, each object has a minimum ranking value and a maximum ranking value, resulting in a score interval $[\tau_{min}, \tau_{max}]$. All objects are sorted in ascending order of τ_{min} (recall that the smaller the score is, the better). Then, the top-k objects are put into a candidate set CS, and a *threshold* is defined as the kth smallest τ_{max} in CS. Next, we check the $(k+i)$th object, where $1 \leq i \leq n$ and n is the total number of objects in the dataset. If the τ_{min} of an object O does not exceed the *threshold*, object O is put into the candidate set, and *threshold* is updated accordingly. Otherwise, the candidate set is returned.

Example 11 In Fig. 11, there are 4 objects O_1, O_2, O_3, and O_4. Each object has a score interval with respect to an RkT^u query Q ($k = 2$ and $Q.L$ is a line segment), computed from Eq. 11. The sorted list of objects is (O_2, O_4, O_3, O_1). We put the top-2 objects O_2 and O_4 into the candidate set and $threshold = O_4.\tau_{max}$. Then, we check O_3 and find that $O_3.\tau_{min} < threshold$. Hence, O_3 is also put into the candidate set, and *threshold* is updated to $O_3.\tau_{max}$. We next check object O_1 and find that $O_1.\tau_{min} > threshold$, so that the candidate set $\{O_2, O_4, O_3\}$ is returned.

To find candidates with the IR-tree or its enhanced versions, we exploit the best-first traversal algorithm. A priority queue is used to keep track of the nodes and objects that have yet to be visited. The values of τ_{min} and $MIND_{ST}$ are used as the keys of objects and nodes, respectively. A candidate set and a *threshold* are maintained. When processing an object, its score interval $[\tau_{min}, \tau_{max}]$ is computed and if $\tau_{min} \leq threshold$ it is added to the candidate set. The

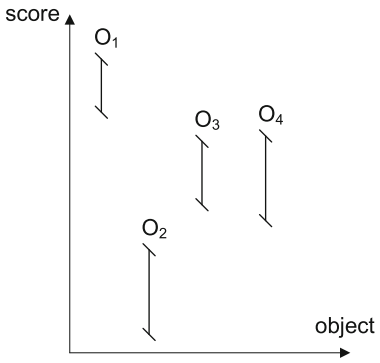

Fig. 11 An example of score intervals

threshold is updated accordingly. When processing a node, we apply the following pruning strategy.

Pruning Strategy. Given an RkT^u query Q, let N be a node in the hybrid index. If $MIND_{ST}(Q, N) > threshold$, then no object in node N can be the result of Q, and thus N is pruned.

Proof Knowing that $MIND_{ST}(Q, N)$ is a lower bound of the ranking values of objects enclosed by node N and *threshold* is the kth minimal τ_{max} in candidate set CS, if $MIND_{ST}(Q, N) > threshold$, there are already k objects in candidate set CS with smaller ranking values than all the objects in N. Hence, node N can be pruned. □

If a node is not pruned, all the entries in the node are added to the priority queue. The termination condition is that the key of the first element in the priority queue is bigger than *threshold*. When this happens, the algorithm stops, and the candidate set is returned. Algorithm 6 shows the pseudo-code of FindCandidates.

6.3 Discussion

No algorithms exist for the processing of RkT^u queries. Hu et al. [19] consider the k range-nearest-neighbor (kRNN) query that requests the union of the k nearest neighbors (kNNs) for every point in a given query range Ω. The result of the kRNN query consists of two kinds of objects, internal kRNN objects that are inside Ω and external kRNN objects that are kNNs for all points on Ω's boundary. They assume the boundary comprises four line segments and propose an algorithm (LNN-Search) to find kNNs for all points on a line segment. The basic idea is to divide the line segment into several subsegments, such that every point in a subsegment has the same kNNs. Those subsegments are determined by the intersections of the line segment and perpendicular bisectors of some pairs of objects.

There are two major differences between the RkT^u query and the kRNN query. One is that the objects inside the query

Algorithm 6 FindCandidates(*Query*, *Index*, k)

```
1: CS ← NewList();
2: threshold ← ∞;
3: Queue ← NewPriorityQueue();
4: Queue.Enqueue(Index.RootNode, 0);
5: while not Queue.IsEmpty() do
6:    Element ← Queue.Dequeue();
7:    if Element.Key > threshold then
8:       Return CS;
9:    if Element is an object then
10:      Compute its score interval [τ_min, τ_max];
11:      if τ_min ≤ threshold then
12:         Add Element to CS;
13:      if |CS| ≥ k then
14:         threshold ← kth min_{O_i ∈ CS}{O_i.τ_max};
15:   else if Element is a leaf node then
16:      for each entry(Object) in leaf node Element do
17:         Queue.Enqueue(Object, τ_min);
18:   else
19:      for each entry(Node) in node Element do
20:         if MIND_ST(Query, Node) ≤ threshold then
21:            Queue.Enqueue(Node, MIND_ST(Query, Node));
22: return CS;
```

region $Q.\Omega$ are not always results. That is because the ranking function $D_{ST}(\cdot, \cdot)$ takes both text relevance and distance into account. An object inside $Q.\Omega$ might have a low text relevance so that it can have a higher value of $D_{ST}(\cdot, \cdot)$ than do some objects outside $Q.\Omega$. The other is that the existing LNN-Search [19] is not applicable to the RkT^u query, since the property of perpendicular bisectors does not hold when using $D_{ST}(\cdot, \cdot)$ as the distance metric.

7 Experimental study

We proceed to evaluate the performance of the algorithms for the processing of the LkT and RkT^u queries using the proposed hybrid indexes and enhancements, including the IR-tree, the DIR-tree, the cluster-enhanced method, the ClusterMBR enhancement, and the TermMBR enhancement, and a recently proposed IR-tree variant.

7.1 Algorithms

In addition to the proposed algorithms, we implemented the two baseline algorithms presented in Sect. 3 as competitors. Since compression techniques for inverted files may reduce the I/O cost of IFO and RIF (while may incur extra computational cost), we favor the IFO and the RIF by excluding the data transfer time from disk to main memory from the reported elapsed time of IFO and RIF reported in all experiments. Note that both baseline algorithms need to compute the text relevancy score for all the objects whose documents contain at least one of the query keywords. In our implementation, we use an accumulator for each object document, and all accumulators are memory resident. However, the hybrid approaches only need accumulators for the objects in a tree node (100 at most in our experiments, to be explained). This means that the baseline algorithms need more memory than the hybrid algorithms.

We also consider an IR-tree variant presented in a recent paper [22]. To differentiate it from our IR-tree, we name it the IRLi-tree, thus combining the name of the index and the first author's name. The ideas underlying the IR-tree and the IRLi-tree are the same: that is, integrating an R-tree with inverted files to prune the search space according to spatial proximity and text relevancy simultaneously. The proposed pseudo document [8] and document summary [22] of a non-leaf node serve to offer an estimate of the text relevancy between the query and the objects in the subtree rooted at the non-leaf node. The two indexes differ only in terms of the storage scheme for the pseudo documents/document summaries of non-leaf nodes: Cong et al. [8] index the pseudo documents in each non-leaf node using a separate inverted file, while Li et al. [22] store the document summaries of all non-leaf nodes in one single inverted file.

Table 8 Dataset properties

Property	DATA
Total # of objects	2,249,727
Average # of unique words per object	429
Total # of unique words in dataset	2,899,175
Total # of words in dataset	965,132,883

7.2 Data and queries

We use a real spatial dataset modeling the roads in California[2] and documents from WEBSPAM-UK2007[3] that consists of a large number of real web documents to generate a dataset DATA by randomly selecting a document for a spatial object. Table 8 lists properties of the dataset.

We generate 4 query sets, in which the number of keywords is 1, 2, 3, and 4, respectively, in the space of DATA. Each set comprises 200 queries. Queries are generated from objects, and we guarantee that no query has an empty result. Specifically, to generate a query, we randomly pick an object in the dataset, and take the location of the object as the query location and randomly choose words from the document of the object as the query keywords.

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

[3] http://barcelona.research.yahoo.net/webspam/datasets/uk2007.

7.3 Setup

All index structures (IR-tree, DIR-tree, enhancements, baselines, IRLi-tree) are disk resident, and the page size is 4 KB. The number of children of a node in the R-tree is computed given the fact that each node occupies a page. This translates to 100 children per node in our implementation.

All algorithms were implemented in Java, and an Intel(R) Xeon(R) CPU E5320@1.86 GHz with 16 GB main memory was used for the experiments. The Java Virtual Machine Heap is set to 10GB. We report the average elapsed time cost of the queries in each query set. Many layers of cache (e.g., disk driver cache, operating system cache, application cache) exist between a Java application and the physical disk. Rather than measuring physical I/Os from the disk using Java, we report the simulated I/O cost using a simulated LRU buffer that contains 5 % of the index.

We study the effect of different parameters and set parameter default values as follows: the number k of requested results is 10; the number of query keywords is 2; parameter α in the ranking function (Eq. 1) is 0.3.

7.4 Tuning experiment

This experiment tunes the number of clusters in the cluster-enhanced method and the parameter β used in the DIR-tree to find good values for the two methods.

Varying the number of clusters

CIR-trees with 5, 10, 15, 20, 30, and 50 clusters are built. We use the k-means algorithm for clustering. The sizes of the resulting CIR-trees are shown in Table 9. The storage increases slightly with the number of clusters.

Figures 12a, b show the performance of top-10 queries. The I/O cost of the CIR-tree comes from the R-tree and the inverted file. As shown in Fig. 12b, the total I/O cost is dominated by the I/O cost of using the inverted file. As the number of clusters increases, the total I/O cost increases since the posting lists in the inverted file become longer. In contrast, the I/O cost from the R-tree decreases when the number of clusters increases. This is because more clusters provide tighter bounds on the text relevancies so that more nodes are pruned in the R-tree. The elapsed time shown in Fig. 12a has the same trend as does the I/O cost from the R-tree, that is, decreasing as the number of clusters increases. This behavior

Table 9 Sizes of different CIR-trees (GB)

# Clusters	5	10	15	20	30	50
Size	77.53	78.56	79.28	80.05	80.35	81.6

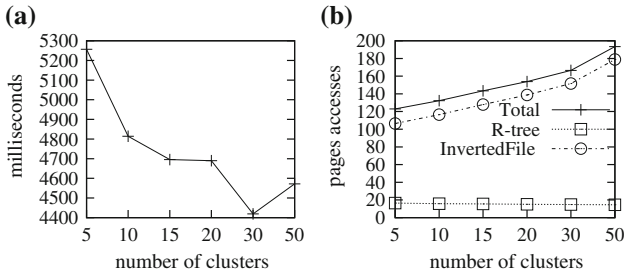

Fig. 12 Varying the number of clusters. **a** Elapsed time. **b** Simulated I/O

occurs because the time needed for loading longer posting lists in the inverted file is less than the time needed for the processing of the nodes pruned by tighter bounds provided by more clusters in the CIR-tree.

However, it is not true that the more clusters, the better the performance always. With too many clusters, for example, 50 clusters, and the time needed for the processing of clusters in non-leaf nodes counteracts the time saved by fewer node visits due to more clusters, and the elapsed time gets worse. In the following experiments, we set the number of clusters to 30.

Varying the importance of document similarity when building DIR-trees

We use the Hamming distance to compute the $DocSim(\cdot, \cdot)$ that occurs in Eq. 4. In order to make the construction of the DIR-tree efficient, when computing Eq. 4, we only consider the top-100 words according to the word frequencies in the whole dataset. We vary the parameter β to build different DIR-trees. Figure 13a, b show the performance of the different DIR-trees. In the extreme case of $\beta = 0$, the tree is actually an IR-tree. In the other extreme case of $\beta = 1$, only document similarity is considered when building the DIR-tree. The performance is best at $\beta = 0.7$. For a specific application, we can find a good parameter value empirically. The value of β is set to 0.7 in subsequent experiments.

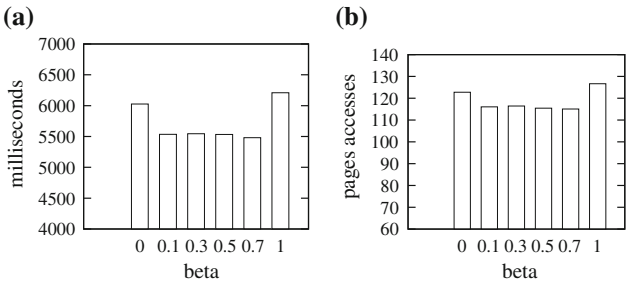

Fig. 13 Varying β. **a** Elapsed time. **b** Simulated I/O

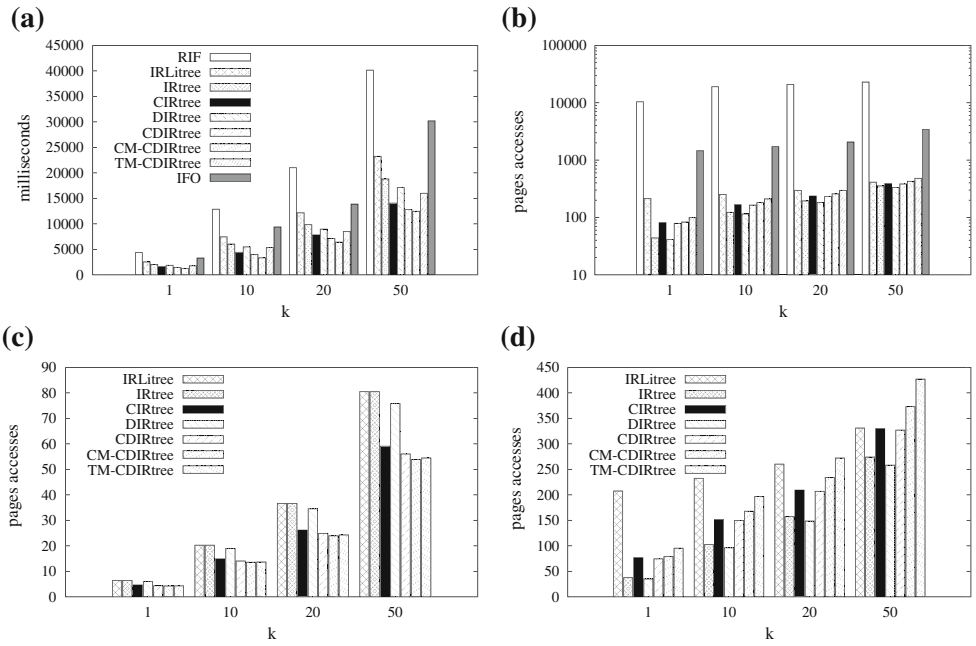

Fig. 14 Varying k in LkT queries. **a** Elapsed time. **b** Simulated I/O. **c** Simulated I/O of the R-tree. **d** Simulated I/O of the inverted files

7.5 Performance evaluation of LkT queries

We apply the cluster-enhanced method to the IR-tree and the DIR-tree, resulting in the CIR-tree and the CDIR-tree. The ClusterMBR and TermMBR enhancements are applied to the CDIR-tree, denoted as the CM-CDIR-tree and TM-CDIR-tree, respectively. Four sets of experiments are carried out. The first evaluates the performance when varying the number k of requested results. The second evaluates the effect of the number of query keywords on performance. The third evaluates the effect of the parameter α in the ranking function. The fourth evaluates the effect of the buffer size. We also report the index size for each approach and study the performance in main memory.

Varying k in LkT queries

Figure 14a, b show the average elapsed time and the average simulated I/O cost. All the hybrid indexes significantly outperform the baseline approaches for all values of k in terms of both metrics.

All the hybrid indexes are based on the R-tree and the inverted files. Figure 14c, d show the average simulated I/O cost of the R-tree and the inverted files, respectively. The I/O costs of the R-trees in the IRLi-tree and the IR-tree are the same, while the I/O cost of the inverted file in the IRLi-tree is notably higher than that of the IR-tree. To see why, recall that both trees construct pseudo documents (document summaries) for non-leaf entries in the R-tree. The IRLi-tree uses a single inverted file to index all pseudo documents and attaches an inverted file to each leaf node in the R-tree, while the IR-tree maintains an inverted file for each node in the R-tree. When processing queries, the posting lists of query keywords to be loaded in the IRLi-tree are longer than those in the IR-tree.

We find that for hybrid indexes, the elapsed time is correlated to the I/O cost of the R-tree. The I/O cost of the R-tree indicates how much the search space is pruned. The higher the I/O cost of the R-tree, the less the searching space is pruned, and vice versa. We also find that the total I/O cost is dominated by the I/O cost of the inverted files. That is because processing one node in the R-tree costs one I/O, but incurring multiple I/Os from the inverted file due to multiple query keywords and long posting lists.

The DIR-tree performs slightly better than the IR-tree, and the cluster enhancement (CIR-tree and CDIR-tree) improves the performance of both the IR-tree and the DIR-tree. This is expected since the DIR-tree accounts for the similarity among documents of objects, while the IR-tree does not. Because the cluster enhancement provides tighter bounds on the text relevancy, the CIR-tree and the CDIR-tree enable more effective pruning of the search space than do the IR-tree and the DIR-tree, as shown in Fig. 14c. The I/O costs of the inverted files in the CIR-tree and the CDIR-tree are higher than those in the IR-tree and the DIR-tree, as shown in Fig. 14d. That is because in the CIR-tree and the CDIR-tree, each non-leaf entry has multiple pseudo documents, which makes posting lists longer. However, the extra I/Os from the inverted files caused by the cluster enhancement do not negatively affect the time saved by its effective pruning. The CM-CDIR-tree improves the performance of the CDIR-tree. The I/O cost of the inverted files in the CM-CDIR-tree is higher

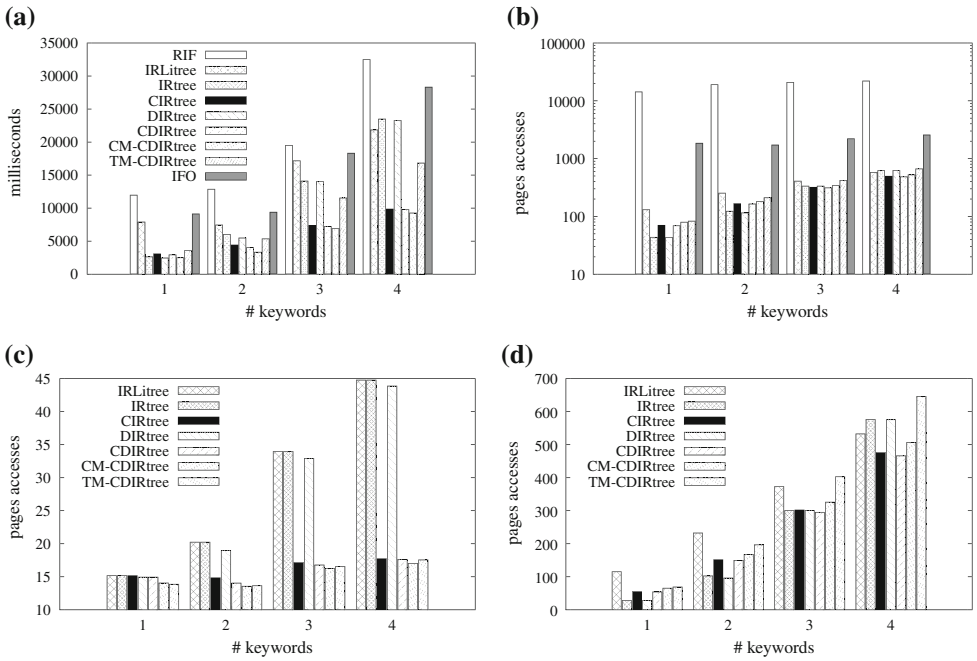

Fig. 15 Varying the number of keywords in LkT queries. **a** Elapsed time. **b** Simulated I/O. **c** Simulated I/O of the R-tree. **d** Simulated I/O of the inverted files

than that of the CDIR-tree, since accessing the MBRs of clusters incurs additional I/Os. However, the tighter bounds provided by cluster MBRs save some I/Os and computation in the tree structure and inverted files, and those savings are not negatively affected by the additional I/Os from cluster MBRs. Hence, the elapsed time of the CM-CDIR-tree is less than that of the CDIR-tree. Although the term MBRs in the TM-CDIR-tree provide more accurate estimate and save some I/O in the tree structure compared with the CDIR-tree (Fig. 14c), accessing the MBRs of terms incurs additional I/O (Fig. 14d) and counteracts the savings provided by accurate estimations. The tradeoff is between the saved time offered by accurate estimate and the additional I/O caused by the data that provides accurate estimate. In this experiment, the CM-CDIR-tree is better than the TM-CDIR-tree.

Varying the number of keywords in LkT queries

Figure 15a, b show that the hybrid indexes outperform the baseline approaches. As the number of keywords increases, the cost of processing queries increases, since more posting lists are loaded from the inverted file as shown in Fig. 15d.

The IR-tree outperforms the IRLi-tree when the number of keywords are 1, 2, and 3. However, the IRLi-tree performs better when the number of keywords is 4, since the I/O cost of the inverted file in the IRLi-tree is lower than that in the IR-tree as shown in Fig. 15d.

This experiment also shows that the DIR-tree performs slightly better than does the IR-tree. We note that when the number of keywords is 1, the cluster enhancement does not improve the IR-tree and the DIR-tree. That is expected since the bound on the text relevancy based on a single keyword is always accurate. In other words, Fig. 15c shows that the IR-tree (DIR-tree) and the CIR-tree (CDIR-tree) do the same amount of search when the number of keywords is 1. However, the I/O costs of the inverted files of the cluster enhancements are higher due to the long posting lists. When the number of keywords is greater than 1, the cluster enhancement can improve the query performance of both the IR-tree and the DIR-tree.

The results for the CM-CDIR-tree and the TM-CDIR-tree are consistent with the results from the previous experiments.

Varying α in LkT queries

Parameter α in Eq. 1 allows users to set their preferences between text relevancy and spatial proximity. Figure 16 shows the results when varying α. A large α means that the spatial distance is more important, while a small α means that the keywords are more important. It is consistent with the previous experiments that the hybrid indexes beat IFO and RIF. The performance of the RIF is improved while the performance of the IFO is getting worse as α increases, since the R-tree in the RIF helps while no spatial index involved in the IFO. As expected, the IR-tree performs better for large α while the DIR-tree performs better for small α. The DIR-tree takes into account document similarity, and the benefit increases when the text relevancy is given higher weight. As in the previous experiment, the cluster enhancement is effective, and the CM-CDIR-tree achieves the best performance.

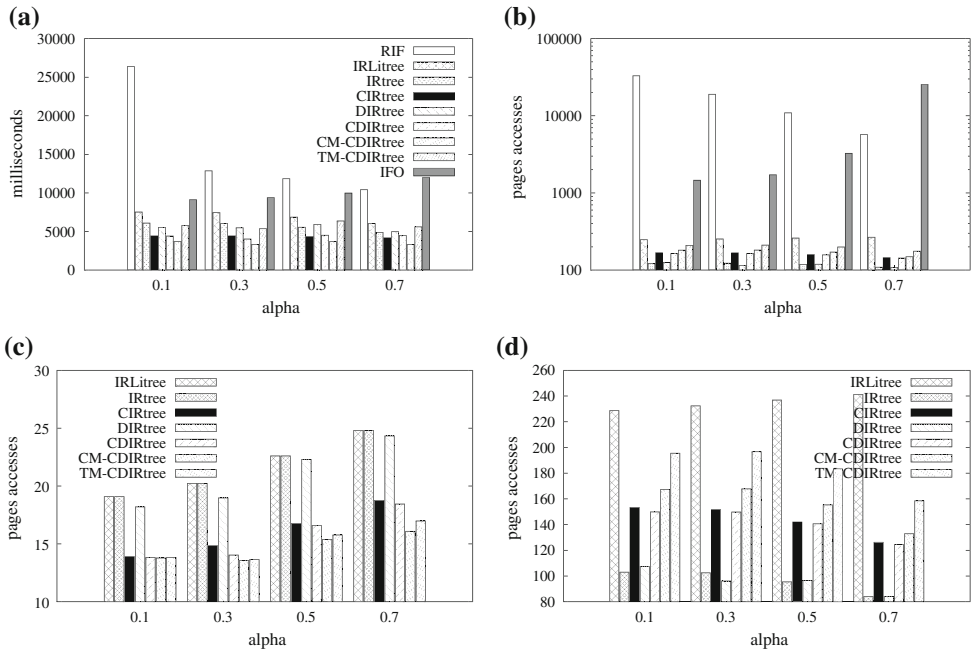

Fig. 16 Varying α in LkT Queries. **a** Elapsed time. **b** Simulated I/O. **c** Simulated I/O of the R-tree. **d** Simulated I/O of the inverted files

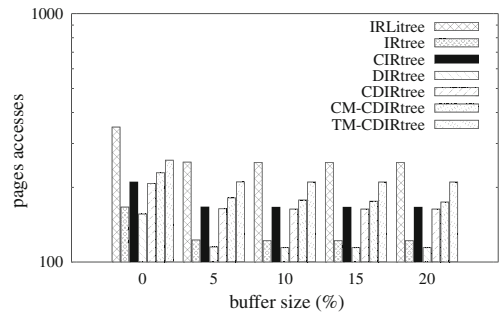

Fig. 17 Varying the buffer size

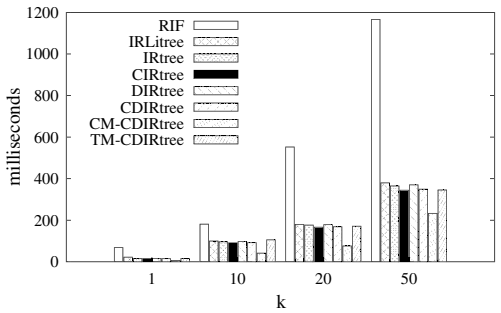

Fig. 18 Evaluation in main memory

Effect of buffering

We use a simulated LRU buffer, and we vary the buffer size from 0 to 20 % of the index size. As shown in Fig. 17, extra buffer space improves the simulated I/O performance of all approaches. As expected, the improvement decreases as the buffer size increases.

Evaluation in main memory

This experiment studies the performance of different approaches in main memory, that is, the whole index structure can fit into main memory. We use a small dataset that combines of a real spatial dataset containing 131,461 objects located in LA streets[4] with five categories of the document dataset 20 Newsgroups.[5] The average elapsed time is reported in Fig. 18. In the main memory setting where no I/O is involved, the hybrid indexes again beat RIF since the hybrid indexes explore a smaller search space. The IR-tree slightly outperforms the IRLi-tree since the IRLi-tree computes the text relevancies of all objects that contain at least one query keyword, while the IR-tree only computes part of them. This experiment also shows that the enhancements improve the performance of the IR-tree with the CM-CDIR-tree being best.

Space requirements

Table 10 shows the total sizes of each index structure. The differences between the RIF and the hybrid indexes in the table show the overhead introduced by the hybrid indexes. As discussed in Sect. 4.1, the inverted file built in RIF is roughly equal to the inverted files in the leaf nodes of the hybrid indexes. The overhead occurs because in the hybrid indexes, each non-leaf node also has an inverted file. The size of the

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

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

Table 10 Index structure sizes (GB)

RIF	IRLi-tree	IR-tree	CIR-tree	DIR-tree	CDIR-tree	CM-CDIR-tree	TM-CDIR-tree
12.77	74.67	75.13	80.35	75.54	80.90	80.94	102.23

IR-tree is slightly bigger than that of the IRLi-tree, since the IR-tree stores an inverted file for each non-leaf node while the IRLi-tree has a single inverted file for all non-leaf nodes. The difference between the IR-tree (resp. DIR-tree) and the CIR-tree (resp. CDIR-tree) shows the storage overhead due to the cluster refined inverted files in the non-leaf nodes in the CIR-tree. Note that the sizes of the inverted files in their leaf nodes are the same. The size of the DIR-tree is similar to that of the IR-tree. Cluster MBRs are stored separately. They only add slightly (0.04 GB) to the storage of the CDIR-tree. The TM-CDIR-tree occupies more space than does the CDIR-tree due to the space occupied by the MBR of each term in the inverted files.

7.6 Performance evaluation of RkT^u queries

There exists no baseline algorithm for RkT^u queries. Thus, we extend the proposed baseline algorithm RIF for LkT queries to process RkT^u queries. The term MBR and the cluster MBR techniques are applied to the CDIR-tree, and the results are compared with the original CDIR-tree and the IR-tree.

Four sets of experiments are carried out. The first evaluates the performance when varying the number k of requested results. The second evaluates the effect of the number of query keywords. The third evaluates the effect of the parameter α in the ranking function. The fourth evaluates the performance when varying the size of the query region. As for LkT queries, parameter β is set to 0.7, and the number of clusters is set to 30. By default, parameter α is set to 0.3, the number of keywords is 2, the size of the query region is set to 0.01 % of the whole space, and $k = 10$.

Varying k in RkT^u queries

Figure 19a, b show the elapsed time and I/O costs of the FindCandidates algorithm with different indexes when varying k. All the four hybrid indexes significantly outperform the baseline approach for all values of k in terms of both elapsed time and I/O cost. Figure 19c, d show the average simulated I/O cost of the R-tree and the inverted files, respectively. This experiment also demonstrates that for hybrid indexes, the elapsed time is correlated with the I/O cost of the R-tree. It is consistent with the study of the LkT query that the CDIR-tree performs better than the IR-tree and that the ClusterMBR technique improves the performance of the CDIR-tree. Figure 19e shows the total elapsed time of processing RkT^u queries. The performance is dominated by the FindCandidates step. The elapsed time increases as k increases. Since the Filtering step incurs no I/O, the total I/O cost is the same as shown in Fig. 19b.

Varying the number of keywords in RkT^u queries

Figure 19f–i show the elapsed time and I/O costs of the FindCandidates algorithm on different indexes when varying the number of keywords. This experiment also illustrates that the hybrid indexes outperform RIF. When the number of keywords is 1, the IR-tree outperforms the CDIR-tree and its variants. The reason is the same as what is explained before. For the other cases, the ClusterMBR technique applied to the CDIR-tree performs the best. Figure 19j shows the total elapsed time of processing RkT^u queries, indicating that the cost of the Filtering step is trivial.

Varying α in RkT^u queries

Figure 19k–o show the performance on different indexes when varying α. The hybrid indexes outperform RIF. It is again found that the CM-CDIR-tree is best.

Varying the size of the query region

Figure 19p–t show the performance on different indexes when varying the size of the query region. Again, this experiment demonstrates that the hybrid indexes outperform RIF significantly and that the enhanced techniques improve performance.

7.7 Summary

The experimental study shows that the proposed hybrid indexes significantly and quite consistently outperform the baseline. It also shows that the proposed enhancements indeed improve the performance of the hybrid indexes. We conclude that applying the ClusterMBR to the CDIR-tree has the best performance for the processing of both LkT and RkT queries in most of cases.

8 Related work

8.1 Nearest neighbor and top-k queries

The processing of k-nearest neighbor queries (kNNs) in spatial databases is a classical subject. Most proposals use index structures to assist in the kNN processing. Perhaps the most influential kNN algorithm is due to Roussopoulos et al. [30].

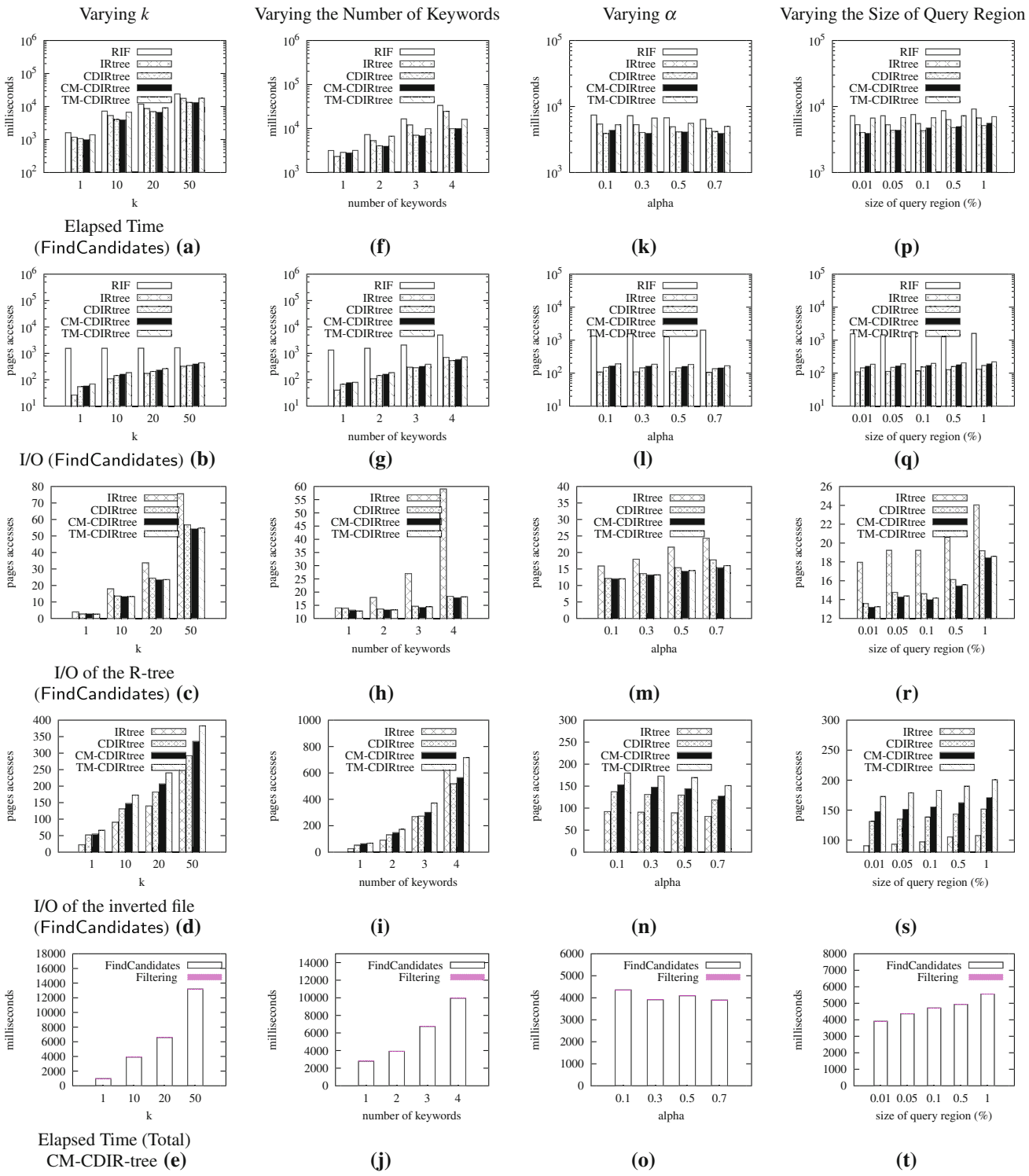

Fig. 19 Evaluating R*k*T queries

In this solution, an R-tree [16] indexes the data points, potential nearest neighbors are maintained in a priority queue, and the tree is traversed according to a number of heuristics. Other methods modify the index structures to better suit the particular problem addressed [20,35]. Hjaltason and Samet [18] propose an incremental nearest neighbor algorithm based on an R*-tree [4].

Our work is also related to top-*k* query processing [5,12]. Fagin et al. [12] propose a class of algorithms known as threshold algorithms. These algorithms, like the ones pro-

posed for information retrieval [2,27], enable efficient computation of aggregate functions over multiple sorted lists. These algorithms can be easily integrated into the leaf nodes in our framework (we need to process all entries in non-leaf nodes, so the threshold algorithm does not apply there); however, using them does not improve performance in our framework. The possible reason is that the posting lists in a leaf node are short (limited by the capacity of a node).

8.2 Text retrieval queries

A variety of retrieval models have been proposed to meet different information retrieval needs, such as probabilistic models, the vector space model, the probabilistic similarity measure often referred to as the Okapi measure (BM25) [29], and language models. The latter represent a relatively new approach, and they offer either the best or competitive performance in many settings [9,28].

Many different types of text indexes have been proposed. The most efficient index structure for text retrieval is the inverted file [39], and many state-of-the-art, large-scale IR systems such as web search engines employ inverted files for ranking-query evaluation. Alternative indexing techniques for text documents also exist, including suffix arrays [3] and signature files [13]. Zobel et al. [40] empirically show that signature files are not competitive with the inverted file for information retrieval queries.

To improve efficiency, a host of works develop effective heuristics for reducing query evaluation costs by reordering the inverted file according to frequency or contribution [2,27]. Other techniques also exist (e.g., [33]) aim to increase query efficiency, and compression techniques exist that aim to reduce storage costs (e.g., [14,26]). These techniques can be applied to our framework; however, they are beyond the scope of this study.

8.3 Location-aware text retrieval queries

Commercial search engines such as Google and Yahoo! have introduced local search services that appear to focus on the retrieval of local content, for example, related to stores and restaurants. However, the algorithms used are not publicized.

Much attention has been given to the problem of extracting geographic information from web pages (e.g., [1,11,25]). The extracted information can be used by search engines. McCurley [25] covers the notion of geo-coding and describes geographic indicators found in web pages, such as zip codes and location names.

A recent study by Zhou et al. [38] tackles the problem of retrieving web documents relevant to a keyword query within a pre-specified spatial region. They propose three approaches based on a loose combination of an inverted file and an R*-tree. The best approach according to their experiments is to build an R*-tree for each distinct keyword on the web pages containing the keyword. As a result, queries with multiple keywords need to access multiple R*-trees, and the results must be intersected. Building a separate R*-tree for each keyword also requires substantial storage.

Our hybrid indexing framework differs substantially from this approach, although both use inverted files and R-trees. Our approach incorporates the inverted file at each node of an R-tree such that both location and text information can be utilized simultaneously to prune the search space at query time, while the approaches of Zhou et al. [38] adopt combinations that require query processing to occur in two stages: One type of indexing is used to filter web document in the first stage, and then another type is employed in the second stage. This is similar in spirit to the baseline approach used in our experiments.

Next, our approach and their approaches target different kinds of queries: while we focus on top-k queries, Zhou et al. aim to retrieve relevant documents within a given geographic region. We know of no way of adapting these approaches to process the top-k queries considered in this paper (without scanning all objects). In contrast, our framework can easily be extended to process the queries considered by Zhou et al.

Another study [7] considers a different type of query that retrieves web pages that contain the query keywords and whose page footprint intersects with the query footprint, where a footprint is an arbitrary, possibly noncontiguous area. They apply spatial filling curve to the inverted index. Vaid et al. [34] also present techniques to combine the output of a text and a spatial index to answer a spatial keyword query in two stages. The aforementioned two differences between our approach and that of Zhou et al. also apply here.

Next, Hariharan et al. [17] address the problem of finding objects containing query keywords within a region. They present a hybrid indexing structure called the KR*-tree that consists of an R*-tree and an inverted file for the nodes of the R*-tree. The nodes of the KR*-tree are virtually augmented with the sets of keywords that appear in the subtrees rooted at the nodes. At query time, the KR*-tree based algorithm finds the nodes that contain the query keywords and then uses these as candidates for subsequent search. This approach suffers from unnecessary overhead when there are many candidates.

Felipe et al. [10] integrate signature files and the R-tree to enable keyword search on spatial data objects that each have a limited number of keywords. This approach needs to load the signature files of all words into memory when a node is visited, which incurs substantial I/O. Signature files are generally inferior to inverted files for general text retrieval [39]. The fact that there is no practical way of using signature files for handling ranked queries [39] renders it infeasible for this approach to support LkT queries that need to compute text relevancy scores (using language models).

Another hybrid indexing structure [37] combines the R*-tree and bitmap indexing to process the m-closest keyword query that returns the spatially closest objects matching m keywords. This approach exhibits the same problems as do signature-file based indexing [10].

Martins et al. [24] compute text relevancy and location proximity independently and then combine the two ranking scores. The baseline algorithms we investigate in this paper appear to be better than this approach.

Khodaei et al. [21] propose the Spatial Keyword Inverted File to handle location-based web searches. They develop a new distance measure called spatial tf-idf to rank relevant documents.

Cao et al. [6] propose the concept of prestige-based relevance to capture both the textual relevance of an object to a query and the effects of nearby objects. The top-k spatial web objects are ranked according to both prestige-based relevance and location proximity.

In a previously published conference paper [8], we introduce LkT queries, and we propose the IR-tree, the DIR-tree, and the cluster-enhanced method. This paper in addition proposes an efficient solution for the processing of a new type of query, called the RkT query, and it proposes and studies two new enhanced techniques for the framework proposed in [8], namely the ClusterMBR, which constructs an MBR for each cluster of each node, and the TermMBR, which constructs an MBR for each word in the inverted files of each node. Section 7.1 compares in depth with the IR-tree proposed by Li et al. [22]. All extensions in this paper to the conference paper [8] go beyond Li et al.'s proposal.

9 Conclusions and future work

This paper proposes a new indexing framework for location-aware top-k text retrieval and region-aware top-k text retrieval. The framework tightly integrates the inverted file for text retrieval and the R-tree for spatial proximity querying in a novel manner. Several hybrid indexing approaches are explored within the framework. The framework encompasses algorithms that utilize the proposed indexes for computing the top-k query, and it is capable of simultaneously taking into account text relevancy and spatial proximity to prune the search space during query processing. Results of empirical studies with an implementation of the framework demonstrate that the paper's proposal is capable of excellent performance.

This work opens to a number of promising directions for future work. First, it is worth adapting existing optimization techniques developed for inverted files (e.g., compression) and R-trees to the paper's setting. Second, the performance of the DIR-tree may be affected by the choice of the document similarity function $DocSim(\cdot, \cdot)$ in Eq. 4. Moreover, the document space has high dimensionality (our dataset has 2+ million distinct words), which makes the construction of the DIR-tree slow. When inserting an object into the DIR-tree, we need to compute the document similarity between the object and some entries in the tree. After finding the node in which to insert the object, we have to update the pseudo documents of its predecessor nodes. When a node overflows, the splitting is also computationally expensive due to the high dimensional document space. We can reduce the document space dimensionality, for example, by considering the top-100 words according to word frequencies. However, it may adversely affect the performance of the DIR-tree. It is worth developing an approach that is able to make the construction of the DIR-tree efficient without hurting its performance. Third, it is of interest to understand how the top-k queries considered can best be processed if the objects are constrained to a spatial network.

References

1. Amitay, E., Har'El, N., Sivan, R., Soffer, A.: Web-a-where: geotagging web content. In: SIGIR, pp. 273–280 (2004)
2. Anh, V.N., de Kretser, O., Moffat, A.: Vector-space ranking with effective early termination. In: SIGIR, pp. 35–42 (2001)
3. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison Wesley, Reading, MA (1999)
4. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. In: SIGMOD, pp. 322–331 (1990)
5. Bruno, N., Gravano, L., Marian, A.: Evaluating top-k queries over web-accessible databases. In: ICDE, pp. 369–380 (2002)
6. Cao, X., Cong, G., Jensen, C.S.: Retrieving top-k prestige-based relevant spatial web objects. In: PVLDB, pp. 373–384 (2010)
7. Chen, Y.-Y., Suel, T., Markowetz, A.: Efficient query processing in geographic web search engines. In: SIGMOD, pp. 277–288 (2006)
8. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. PVLDB **2**(1), 337–348 (2009)
9. Cong, G., Wang, L., Lin, C.-Y., Song, Y.-I., Sun, Y.: Finding question-answer pairs from online forums. In: SIGIR, pp. 467–474 (2008)
10. De Felipe, I., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: ICDE, pp. 656–665 (2008)
11. Ding, J., Gravano, L., Shivakumar, N.: Computing geographical scopes of web resources. In: VLDB, pp. 545–556 (2000)
12. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. **66**(4), 614–656 (2003)
13. Faloutsos, C., Christodoulakis, S.: Signature files: an access method for documents and its analytical performance evaluation. ACM TODS **2**(4), 267–288 (1984)
14. Faloutsos, C., Jagadish, H.V.: Hybrid index organizations for text databases. In: EDBT, pp. 310–327 (1992)
15. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., San Francisco, CA (1979)
16. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: SIGMOD, pp. 47–57 (1984)
17. Hariharan, R., Hore, B., Li, C., Mehrotra, S.: Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In: SSDBM, p. 16 (2007)

18. Hjaltason, G.R., Samet, H.: Distance browsing in spatial databases. ACM Trans. Database Syst. **24**(2), 265–318 (1999)
19. Hu, H., Lee, D.L.: Range nearest-neighbor query. IEEE Trans. Knowl. Data Eng. **18**(1), 78–91 (2006)
20. Katayama, N., Satoh, S.: The SR-tree: an index structure for high-dimensional nearest neighbor queries. In: SIGMOD, pp. 369–380 (1997)
21. Khodaei, A., Shahabi, C., Li, C.: Hybrid indexing and seamless ranking of spatial and textual features of web documents. In: DEXA, pp. 450–466 (2010)
22. Li, Z., Lee, K.C.K., Zheng, B., Lee, W.-C., Lee, D., Wang, X.: IR-tree: an efficient index for geographic document search. IEEE Trans. Knowl. Data Eng. **23**(4), 585–599 (2011)
23. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982)
24. Martins, B., Silva, M.J., Andrade, L.: Indexing and ranking in geo-IR systems. In: GIR, pp. 31–34 (2005)
25. McCurley, K.S.: Geospatial mapping and navigation of the web. In: WWW, pp. 221–229 (2001)
26. Moffat, A., Zobel, J.: Coding for compression in full-text retrieval systems. In: Data Compression Conference, pp. 72–81 (1992)
27. Persin, M., Zobel, J., Sacks-Davis, R.: Filtered document retrieval with frequency-sorted indexes. J. Am. Soc. Inf. Sci. **47**(10), 749–764 (1996)
28. Ponte, J.M., Croft, W.B.: A language modeling approach to information retrieval. In: SIGIR, pp. 275–281 (1998)
29. Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M.M., Gatford, M.: Okapi at TREC-3. In: TREC (1994)
30. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: SIGMOD, pp. 71–79 (1995)
31. Sanderson, M., Kohler, J.: Analyzing geographic queries. In: SIGIR Workshop on Geographic Information Retrieval (2004)
32. Schnitzer, B., Leutenegger, S.: Master-client R-trees: a new parallel R-tree architecture. In: SSDBM, pp. 68–77 (1999)
33. Strohman, T., Turtle, H., Croft, W.B.: Optimization strategies for complex queries. In: SIGIR, pp. 219–225 (2005)
34. Vaid, S., Jones, C.B., Joho, H., Sanderson, M.: Spatio-textual indexing for geographical search on the web. In: SSTD, pp. 218–235 (2005)
35. White, D.A., Jain, R.: Similarity indexing with the SS-tree. In: ICDE, pp. 516–523 (1996)
36. Zhai, C., Lafferty, J.: A study of smoothing methods for language models applied to information retrieval. ACM TOIS **22**(2), 179–214 (2004)
37. Zhang, D., Chee, Y.M., Mondal, A., Tung, A.K.H., Kitsuregawa, M.: Keyword search in spatial databases: Towards searching by document. In: ICDE, pp. 688–699 (2009)
38. Zhou, Y., Xie, X., Wang, C., Gong, Y., Ma, W.-Y.: Hybrid index structures for location-based web search. In: CIKM, pp. 155–162 (2005)
39. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Comput. Surv. **38**(2), 1–56 (2006)
40. Zobel, J., Moffat, A., Ramamohanarao, K.: Inverted files versus signature files for text indexing. ACM TODS **23**(4), 453–490 (1998)