Efficient Algorithms and Cost Models for Reverse Spatial-Keyword k-Nearest Neighbor Search

YING LU and JIAHENG LU, Renmin University of China
GAO CONG, Nanyang Technological University
WEI WU, Institute for Infocomm Research, Singapore
CYRUS SHAHABI, University of Southern California

Geographic objects associated with descriptive texts are becoming prevalent, justifying the need for spatial keyword queries that consider both locations and textual descriptions of the objects. Specifically, the relevance of an object to a query is measured by *spatial-textual similarity* that is based on both spatial proximity and textual similarity. In this article, we introduce the Reverse Spatial-Keyword k-Nearest Neighbor (RSKkNN) query, which finds those objects that have the query as one of their k nearest spatial-textual objects. The RSKkNN queries have numerous applications in online maps and GIS decision support systems.

To answer RSKkNN queries efficiently, we propose a hybrid index tree, called IUR-tree (Intersection-Union R-Tree) that effectively combines location proximity with textual similarity. Subsequently, we design a branch-and-bound search algorithm based on the IUR-tree. To accelerate the query processing, we improve IUR-tree by leveraging the distribution of textual description, leading to some variants of the IUR-tree called clustered IUR-tree (CIUR-tree) and combined clustered IUR-tree (C^{2}IUR-tree), for each of which we develop optimized algorithms. We also provide a theoretical cost model to analyze the efficiency of our algorithms. Our empirical studies show that the proposed algorithms are efficient and scalable.

Categories and Subject Descriptors: H.2.8 [**Database applications**]: Spatial databases and GIS

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Reverse k-Nearest Neighbor queries, Spatial-Keyword Query, Performance Analysis

ACM Reference Format:
Ying Lu, Jiaheng Lu, Gao Cong, Wei Wu, and Cyrus Shahabi. 2014. Efficient Algorithms and Cost Models for Reverse Spatial-Keyword k-Nearest Neighbor Search. ACM Trans. Datab. Syst. V, N, Article 1 (January YYYY), 42 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. Introduction

With the advent of Web 2.0, many Web objects are associated with both textual contents and locations. For example, more than 7 million tweets per day are geotagged, or many review websites such as Yelp have both location and textual information, such as "`Seafood buffet promotion`" or "`Japanese sushi takeaway`". This phenomena gives rise to spatial keyword queries that can search for objects in both keywords and location spaces.

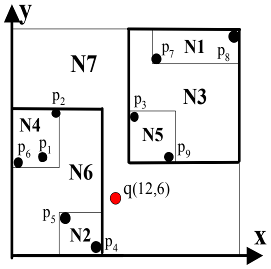
This research is partially supported by 973 Program of China (Project No. 2012CB316205), NSF China (No: 61170011), RUC Research Funds (No.11XNJ003), Singapore MOE AcRF Tier 2 Grant (ARC30/12) and Tier 1 Grant (RG66/12).

Author's addresses: Y. Lu, (current address) Department of Computer Science, University of Southern California, Los Angeles, CA, 90089; email:lvying603@gmail.com. The bulk of the research of Y. Lu was done at Renmin University, and she is currently affiliated with University of Southern California. J. Lu (corresponding author), DEKE, MOE and School of Information, Renmin University of China, Beijing, China, 100872; email:jiahenglu@ruc.edu.cn. G. Cong, School of Computer Engineering, Nanyang Technological University, Singapore, 639798; email:gaocong@ntu.edu.sg. W. Wu, Data Analytics Department, Institute for Infocomm Research, Singapore 138632; email:wwu@i2r.a-star.edu.sg. C. Shahabi, Department of Computer Science, University of Southern California, Los Angeles, CA, 90089; email:shahabi@usc.edu.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

Towards this end, we introduce a new type of spatial-keyword query, dubbed Reverse Spatial-Keyword k-Nearest Neighbor (RSKkNN), which is a type of RkNN queries for finding objects whose k-nearest neighbors (kNN) include the query. RkNN has received considerable attention in the recent decades due to its importance in several applications involving decision support [Korn and Muthukrishnan 2000; Kang et al. 2007; Wu et al. 2008a], resource allocation [Cheema et al. 2009], profile-based marketing [Emrich et al. 2010], etc. Among many of these applications, the RkNN is mainly used to discover *influence sets*. An influence set is a set of objects in a dataset that are highly influenced by the query object. For example, existing stores may be "*influenced*" by a new store outlet since their customers may be closer to the new store and they may be attracted by the new store. To illustrate, consider the example in Figure 1. The points $p_1 \cdots p_9$ in Fig. 1(a) are existing stores in a region, and q is a new store (the rectangles N1$\cdots$N7 in Fig. 1(a) are MBRs that will be explained later in Section 5). Assuming k=2, the results of the RkNN query for point q are $\{p_4, p_5, p_9\}$, as q is the top-2 spatial nearest neighbor of p_4, p_5 and p_9.

In previous studies [Stanoi et al. 2000; Tao et al. 2004; Achtert et al. 2006; Wu et al. 2008b], spatial distance is usually considered as the sole influence factor. However, in real applications, distance alone may not be sufficient to characterize the influence between two objects. For example, two objects (e.g., restaurants) are more likely to influence each other if their textual descriptions (e.g., seafood buffet lunch including crab and shrimp) are similar. Therefore, in this article, we incorporate textual similarity in RkNN, and study a new type of the RkNN problem, named Reverse Spatial-Keyword k Nearest Neighbor (RSKkNN), where both spatial distance and textual similarity are considered. The RSKkNN query finds the objects that have the query object as one of their k most spatial-textual nearest objects. Recall Figure 1, which illustrates the difference between our proposed RSKkNN query and the conventional RkNN query. Points $p_1 \cdots p_9$ in Fig. 1(a) are existing stores in a region, and q is a newly opened store. The textual description of each store is given in Fig. 1(b), where the weight of each word can be calculated using the TF-IDF measure [Salton 1988]. An RSKkNN query with q as the query object finds the existing stores that will be influenced most by q considering both the spatial proximity and the textual similarity. For example, suppose k=2, the results of the traditional RkNN query are $\{p_4, p_5, p_9\}$, while the results of our RSTkNN query will be $\{p_1, p_4, p_5, p_9\}$. Note p_1 becomes an answer since the textual description of p_1 is similar to that of q, and q is a top-2 spatial-textual nearest neighbor when spatial proximity and textual similarity are considered. However, q is not a 2-NN of p_1 when spatial distance alone is considered.



(a) Distribution of branch stores

	x	y	vectors	stationery	sportswear	pan	diaper	camera	laptop
p_1	3	12	ObjVct1	8	8	0	0	0	0
p_2	4	16	ObjVct2	1	1	8	8	4	4
p_3	14	15	ObjVct3	1	1	4	4	1	1
p_4	11	0	ObjVct4	7	7	1	1	0	0
p_5	6	5	ObjVct5	4	4	1	1	0	0
p_6	0	11	ObjVct6	1	1	7	7	0	0
p_7	18	20	ObjVct7	0	0	0	0	8	8
p_8	25	22	ObjVct8	1	1	0	0	7	7
p_9	19	10	ObjVct9	0	0	1	1	4	4
q	12	6	ObjVctQ	8	8	0	0	0	0

(b) Locations and products of branch stores in (a)

Fig. 1. An example of RSKkNN queries

RSKkNN queries have many applications ranging from map-based Web search to GIS decision support. For example, a shopping mall can use RSKkNN queries to find potential customers whose profiles are relevant to the products of the shopping mall and whose locations are close to this shopping mall. As another example, a person who wants to buy/rent a house would describe her/his desired house with both location and textual description that specifies the amenities (s)he wants.

The RSKkNN query can help landlords find the potential buyers/renters who may be interested in their houses based on the location and description of the houses.

Unfortunately, taking into account the textual relevance in RSKkNN will pose great challenges to the existing techniques for processing conventional RkNNs (without considering textual relevance), and render them inapplicable to process RSKkNN queries. In particular, an attempt to solve the RSKkNN problem using the existing methods is to map the keywords to feature dimensions, and use the existing techniques for conventional RkNN queries. Unfortunately, this simple solution has the following limitations: Existing solutions for RkNN queries are based on the ℓp norm metric space, which is suitable to compute the similarity for dense dataset (e.g., location points) but not for the high dimensional and sparse dataset [Tan et al. 2005]. However, the spatial-keyword objects in our problem, which is the fusion of geographical coordinates of point data and the textual descriptions, can be both high dimensional and sparse. Thus, most of existing algorithms based on ℓp norm metric space are *not effective* for answering RSKkNN queries. Even if we can use the ℓp norm metric to measure the textual similarity, they might still suffer from a severe efficiency problem (a.k.a. "*curse of dimensionality*") ([Stanoi et al. 2000; Stanoi et al. 2001; Tao et al. 2004; Wu et al. 2008b; Cheema et al. 2011]). Finally, the work in [Singh et al. 2003] proposes an efficient approach to answer RkNN queries in high dimension. Unfortunately, their algorithm can only provide approximate answers in high dimension. Note that our problem requires efficient algorithms to provide *exact* answers.

Therefore, to process an RSKkNN query accurately and efficiently, in this article, we propose a series of carefully designed solutions and optimizations. In particular, we first give a formal definition of RSKkNN queries, which combines the Euclidean distance for spatial data and the extended Jaccard similarity for textual data. We then propose an effective hybrid indexing structure called Intersection-Union-R tree (IUR-tree) that stores both spatial and textual information. We develop an efficient branch-and-bound algorithm to process RSKkNN queries based IUR-trees by effectively computing spatial-textual similarities between index nodes. We carefully design the upper and lower bounds on the similarity between nodes to avoid the access of irrelevant index nodes, thus saving I/O costs. In addition, as the main theoretical contribution of this article, we propose a cost model and analyze the performance of our algorithm theoretically based on IUR-trees. We are not aware of any existing cost model with the fusion of location proximity and textual similarity.

To further optimize our algorithm, we then propose two enhanced hybrid indexes, namely Clustered IUR-tree (i.e. CIUR-tree) and Combined CIUR-tree (i.e. C^{2}IUR-tree), which enriches the entry contents of R-trees by adding cluster information of texts and changes the method of R-tree construction. Algorithms based on CIUR-tree and C^{2}IUR-tree are also proposed, by leveraging the cluster information to change the order of node access during the traversal of trees to speedup the prcessing. Finally, results of empirical studies with implementations of all the proposed techniques demonstrate the scalability and efficiency of our indexes and algorithms.

Outline of the article. The article is structured as follows. Section 2 defines our research problem. In Sections 3 and 4, we extensively survey the related work and show baseline algorithms, respectively. The IUR-tree index and the RSKkNN algorithm are presented in Sections 5 and 6, respectively. In particular, we develop a cost model and analyze the complexity of our algorithm in Section 6.3. Sections 7 is dedicated to the CIUR-tree and the C^{2}IUR-tree. Section 8 reports on the experimental results and finally Section 9 concludes this article.

2. Problem Definition

We treat the textual content of a Web object as a bag of weighted words. Formally, a document is defined as $\{<d_i,w_i>\}$, $i = 1\cdots m$, where w_i is the weight of word d_i. The weight can be computed by the well-known TF-IDF scheme [Salton 1988].

Let P be a universal Web object set. Each object $p \in P$ is defined as a pair $(p.loc, p.vct)$, where $p.loc$ represents the spatial location information and $p.vct$ is the associated text represented in vector space model. We define RSKkNN query as follows. Given a set of objects P and a query point q (loc,vct), RSKkNN(q, k, P) finds all objects in the database that have the query point q as one of the

k most “similar” neighbors among all points in P, where the similarity metric combines the spatial distance and textual similarity. Following the existing work [I.D.Felipe et al. 2008; Cong et al. 2009], we define a similarity metric, called **spatial-textual similarity**[1], in Eqn(1), where parameter $\alpha \in [0, 1]$ is used to adjust the importance of the spatial proximity and the textual similarity factors. Note that users can adjust the parameter α at the query time.

$$SimST(p_1, p_2) = \alpha * SimS(p_1.loc, p_2.loc) + \\ (1 - \alpha) * SimT(p_1.vct, p_2.vct) \tag{1}$$

$$SimS(p_1.loc, p_2.loc) = 1 - \frac{dist(p_1.loc, p_2.loc) - \varphi_s}{\psi_s - \varphi_s} \tag{2}$$

$$SimT(p_1.vct, p_2.vct) = \frac{EJ(p_1.vct, p_2.vct) - \varphi_t}{\psi_t - \varphi_t} \tag{3}$$

As shown in Eqn(2), the spatial proximity $SimS(.,.)$ of objects $p_1, p_2 \in P$ describes the spatial closeness based on the Euclidean distance $dist(p_1.loc,\ p_2.loc)$. In Eqn(2), φ_s and ψ_s denote the minimum and maximum distance of pairs of distinct objects in P. They are used to normalize the spatial distance to the range $[0, 1]$. The textual similarity $SimT(.,.)$ of objects $p_1, p_2 \in P$ is shown in Eqn(3). Similarly, φ_t and ψ_t are the minimum and maximum textual similarity of pairs of distinct objects in the dataset, respectively. Specifically, $EJ(p_1.vct, p_2.vct)$ is the Extended Jaccard [Tan et al. 2005], which is widely used in textual similarity computing, as shown in Eqn(4).

$$EJ(\vec{v}, \vec{v'}) = \frac{\sum_{j=1}^{n} w_j \times w'_j}{\sum_{j=1}^{n} w_j^2 + \sum_{j=1}^{n} {w'_j}^2 - \sum_{j=1}^{n} w_j \times w'_j}, \\ where\ \vec{v} =< w_1, \cdots, w_n >,\ \vec{v'} =< w'_1, \cdots, w'_n > \tag{4}$$

Alternatively, the textual similarity can also be measured by other distance measures such as cosine similarity, Euclidean similarity, Pearson Correlation Coefficient (PCC) [Strehl et al. 2000], averaged Kullbak-Leibler divergence (KL divergence) [Kullback and Leibler 1951], or the categorical similarity measures in [Boriah et al. 2008]. For example, cosine similarity between two textual vectors $\vec{v}$ and $\vec{v'}$ is given in Eqn. (5), where cosine similarity is defined by the cosine of the angle between two vectors independent from the length difference of the two vectors. Therefore, cosine is translation variant but scale invariant, whereas Euclidean similarity is translation invariant but scale variant. Extended Jaccard combines both aspects of direction and length differences of the two vectors. Previous studies [Huang 2008; Lee and Welsh 2005; Haveliwala et al. 2002; Strehl et al. 2000], which extensively compare various distance measures for text, show that there is no similarity measure that outperforms other measures in all cases. In fact, their difference in many applications is not significant. In this article, we present our new algorithms using the extended Jaccard, but it is important to note that our algorithm is not specific to the extended Jaccard and we will discuss how to extend our method to other similarity measure such as cosine similarity.

$$Cosine(\vec{v}, \vec{v'}) = \frac{\sum_{j=1}^{n} w_j \times w'_j}{\sqrt{\sum_{j=1}^{n} w_j^2} * \sqrt{\sum_{j=1}^{n} {w'_j}^2}}, \\ where\ \vec{v} =< w_1, \cdots, w_n >,\ \vec{v'} =< w'_1, \cdots, w'_n > \tag{5}$$

Formally, given a query object q=(loc, vct), an object $q{\in}P$ is one of k most similar objects with p, denoted by $q{\in}\ SKkNN(p, k, P)$ if and only if it satisfies the condition:

$$|\{o \in P | SimST(o, p) \geq SimST(q, p)\}| < k$$

[1] Hereafter, “spatial-textual similarity” is also called “similarity” for short.

Given a query q, RSKkNN query retrieves objects whose k most similar objects include q. It is formally defined as:

$$RSKkNN(q,k,P) = \{p \in P | q \in SKkNN(p,k,P)\} \tag{6}$$

For example, in Fig. 1, given a query $q(12,6)$ whose textual vector is $<$($stationery$,8), ($sportwear$,8)$>$, and k=2, α=0.6, then $RSKkNN(q,k,P)$=$\{p_1$, p_4, p_5, $p_9\}$. Note that p_1 is an answer due to the high textual similarity between p_1 and q.

3. Related Work

In this section, we review the existing studies on reverse kNN queries, and analyze why they are not applicable to process RSKkNN queries. We also extensively survey related works on spatial keyword queries and cost models on R-tree family index structures.

3.1. Reverse k Nearest Neighbor Queries

Reverse k Nearest Neighbor (RkNN) queries have applications in decision support systems, profile based marketing, data streaming, document databases, and bio-informatics. There exist a host of works on RkNN queries. The existing approaches for RkNN can be grouped into the following two categories.

1) The first class of solutions is based on *pre-computation*. In particular, [Korn and Muthukrishnan 2000] shows a pre-processing based algorithm for answering RNN (i.e., k=1) queries. In the pre-processing stage, each object's nearest neighbor is found, and a circle centered at the object with distance to its nearest neighbor as radius is created. Given a query node q, if q appears in the circle of node n, then n is one of answers. [Lin et al. 2003] proposes an index structure called RDNN-tree (R-tree containing Distance of Nearest Neighbors) to facilitate the processing of RNN queries. Those pre-computing methods naturally extends to $k > 1$. However, they cannot work for RSKkNN queries, since the value of k in an RSKkNN query is given online at the query time. It is impractical to pre-compute each object's k spatial-textual nearest neighbors for all possible values of k.

2) The second class of solutions is based on ℓp norm metric space [Stanoi et al. 2000; Stanoi et al. 2001; Tao et al. 2004; Wu et al. 2008b; Cheema et al. 2011; Achtert et al. 2009]. Stanoi et al. [Stanoi et al. 2000] propose an algorithm for processing an RNN query that does not require the pre-computation of the nearest neighbor circles. The idea is to split the data space centered at query point into six regions of 60° each. The algorithm finds the query point's k nearest neighbors in each of the six regions and examines whether they are the query's RkNN by checking whether the query point is one of their k nearest neighbors. This algorithm reduces an RkNN query to six conditional kNN queries and $6 * k$ kNN queries.

A variation of RkNN is called the bichromatic RNN query, for which Stanoi et al. further propose a Voronoi based algorithm [Stanoi et al. 2001]. They observe that a bichromatic RNN query's influence region is the query point's Voronoi cell. Thus, they design a method that comprises three steps: *approximate*, *refine* and *filter*. Their methods cannot be extended to process RSKkNN queries as the textual space is a high dimensional space and the cardinality of regions increases exponentially in terms of the number of n dimensions (i.e., 3^n-1 for n dimensions [Singh et al. 2003]).

[Tao et al. 2004], [Wu et al. 2008b], and [Cheema et al. 2011] propose bisector-based solutions. These solutions exploit the following geometric property of a bisector: a bisector between two points p and q (query point) divides the data space into two half-planes, and p is closer than q to the points in the half-plane that contains p. Hence, if an object is contained in more than k such half-planes, there exist more than k objects that are closer to the object than the query point, and therefore the object cannot be a result of the RkNN query q.

Achtert et al. [Achtert et al. 2009] propose an algorithm that processes RkNN queries by estimating the lower bound and the upper bound of an object's (and an index entry's) distance to its kth nearest neighbor. If the distance between a query point and an object is larger than the estimated upper bound of the kNN distance, the object is pruned. On the other hand, if the distance between the query point and an object is shorter than the estimated lower bound of the kNN distance, the object belongs to the result set. As more objects are retrieved, their upper bound and lower bound of kNN distance becomes tighter, and finally all objects are either pruned out or included in the final result set.

To sum up for the second class of solutions, the previous studies are based on ℓp norm metric space and they exploit geometric properties to facilitate the processing of RkNN queries. Unfortu-rally, they cannot address the RSKkNN queries, which combines the RkNN and textual similarity search. This is because with textual information, the geometric properties are lost. Further, the ℓp norm metric space is not suitable for computing the similarity between textual descriptions as their vector representations are high dimensional and sparse [Tan et al. 2005].

3.2. Spatial Keyword Queries

Queries on spatial objects associated with textual information are closely related to the RSKkNN query. *Top-k spatial keyword query*, proposed in [I.D.Felipe et al. 2008], is a combination of a top-k spatial query and a keyword query. The result of a top-k query is a list of the top-k objects ranked according to a ranking function that considers both distance and text relevance. *Location-aware top-k text retrieval (LkT) query* [Cong et al. 2009] is similar to *top-k spatial keyword query*. In an LkT query, the text relevancy score can be computed using the information retrieval models (e.g., TF-IDF model). Cong et al. [Cong et al. 2009] propose an indexing structure called IR-tree for processing the LkT query. During the processing of an LkT query, the minimum spatial-textual similarity between query and index node is computed to guide the search for the top k spatial-textual relevant objects. In addition, the LkPT query [Cao et al. 2010] extends the LkT query by taking “prestige” into account in text relevance computation where a relevant object with nearby objects that are also relevant is prestigious and thus preferred. These queries are different from the RSKkNN queries, which can be considered as the “reverse” version of the *spatial keyword query*.

Indexing Structures for Spatial Keyword Queries Several indexing structures have been proposed to facilitate the processing of spatial keyword queries ([Vaid et al. 2005] [Zhou et al. 2005] [I.D.Felipe et al. 2008] [Cong et al. 2009] [Zhang et al. 2009] [Li et al. 2011]).

Vaid et al. [Vaid et al. 2005] propose two spatial-textual indexing schemes based on grid indexing and inverted file. Zhou et al. [Zhou et al. 2005] consider three hybrid index structures that integrate inverted files and R*-tree in different ways: (i) inverted file and R*-tree index, (ii) first inverted file then R*-tree, (iii) first R*-tree then inverted file. It is shown that the second scheme works the best for location-based web search.

The IR-tree [Cong et al. 2009] structure augments an R-tree node with inverted lists, which is suitable for location-aware top-k text retrieval (LkT) queries that load posting lists only for the query keywords. Li et al. [Li et al. 2011] presents an index structure, which is also called IR-tree. To distinguish the two IR-trees, we refer to the IR-tree [Li et al. 2011] as the Li-IR-tree, and that in [Cong et al. 2009] as the Cong-IR-tree. The difference between the Cong-IR-tree and the Li-IR-tree is that the Cong-IR-tree stores the inverted files for each node separately while the Li-IR-tree stores one integrated inverted file for all the nodes. More specifically, the posting list for each term in the Li-IR-tree corresponds to the concatenation of the posting lists of all the nodes of the Cong-IR-tree. Note that with the RSKkNN queries, we need the information about all the words in an entry to estimate similarity between entries, which are provided in neither Cong-IR-tree nor Li-IR-tree. In [Khodaei et al. 2012], Khodaei et. al also combine a spatial distance measure with a textual distance measure. They use TF/IDF for text and then devise their own spatial similarity measure to make it consistent with the fundamentals of TF/IDF. This way they could use a single inverted-file index structure for both the textual and spatial features of the objects. However, they focuse on the top-k queries only.

3.3. Reverse top-k queries

Recent work on *reverse top-k queries* [Vlachou et al. 2010] is also relevant to our work. *Reverse top-k query* is a “reverse” version of top-k query and it also finds the objects that are influenced by a query object. Given a set of user preferences and a set of objects, a *reverse top-k query* finds an object for a set of users for whom the object is one of their top-k objects. Reverse top-k queries and RSKkNN queries are different in the following two aspects. First, RSKkNN queries consider spatial proximity while reverse top-k queries do not. Second, RSKkNN queries work on objects associated with location information and text description while reverse top-k queries work on objects and user preferences described by a set of numerical attribute values. Due to these fundamental differences, techniques developed for reverse top-k queries are not applicable to RSKkNN queries.

3.4. Cost models on R-tree family index structures

In this article, for the first time (to the best of our knowledge), we propose a cost model and theoretical analysis for a query that considers the fusion of both location proximity and textual similarity. Hence, in this section, we extensively review performance analysis on the R-tree family index structures that has been studied for various spatial queries in the past decades. The previous studies on cost model can be divided into five groups: (i) for range queries and window queries; (ii) for k nearest neighbor queries; (iii) for spatial join queries; and (iv) for continuous queries; and (v) for reverse k nearest neighbor queries in L_p-norm space.

3.4.1. Cost analysis for range queries and window queries. There has been a large body of work [Pagel et al. 1993; Kamel and Faloutsos 1993; Faloutsos and Kamel 1994; Theodoridis and Sellis 1996; Papadopoulos and Manolopoulos 1997] that studies the cost models for predicting the performance of R-trees on the execution of range (or window) queries. Specifically, Faloutsos et al. [Faloutsos et al. 1987] present a model that estimates the performance of R-trees and R$^+$-trees assuming uniform distribution of the data. Kamel and Faloutsos [Kamel and Faloutsos 1993] and Pagel et al. [Pagel et al. 1993] independently estimate the number of disk accesses for window queries, assuming that the MBR of each node of the R-tree is already given. Based on the work [Kamel and Faloutsos 1993; Pagel et al. 1993], Foloutsos and Kamel [Faloutsos and Kamel 1994] use a property of the dataset called *fractal dimension* to model R-tree performance for non-uniform distribution. However the model [Faloutsos and Kamel 1994] is applicable only to point datasets. Theodoridis et al. [Theodoridis and Sellis 1996] propose an analytical model which predicts the performance of R-trees for range queries based on the *density* property of the dataset without assuming uniform data distribution. The model works for both point and non-point datasets.

3.4.2. Cost analysis for k nearest neighbor queries. Papadopoulos et al. [Papadopoulos and Manolopoulos 1997] provide lower and upper bounds of the nearest neighbor query performance on R-trees for the L_2 norm metric. Korn et al. [Korn et al. 2001] extends the work [Papadopoulos and Manolopoulos 1997] for k-nearest neighbor queries with arbitrary parameter k. However, the bounds [Korn et al. 2001] become excessively loose when the dimensionality of k increases, rendering it impractical for high dimensional data. Berchtold et al. [Berchtold et al. 1997] present a cost model for query processing in high-dimensional data spaces, and Tao et al. [Tao et al. 2004] propose a cost model for kNN queries in low and medium dimension spaces.

3.4.3. Cost analysis for spatial join queries. To the best of our knowledge, the work by Huang et al. [Huang et al. 1997] is the first attempt to provide a formula to predict the efficiency for spatial join queries. Theodoridis et al. [Theodoridis et al. 2000] present a model that predicts the performance of R-tree-based index structures for selection queries and an extension of this model for supporting join queries. Moreover, an analytical model and a performance study of the similarity join operation is given in [Bohm and Kriegel 2001]. Furthermore, [Corral et al. 2006] gives a cost model for the k-Closest-Pairs query (a type of distance join in spatial databases), which discovers the k pairs of objects formed from two different datasets with the k smallest distances.

3.4.4. Cost analysis for continuous queries. [Tao and Papadias 2003] presents three cost models for continuous queries, including continuous window queries, continuous k nearest neighbor queries and continuous spatial joins, based on TPR-tree [Saltenis et al. 2000]. The three models are based on a general framework for transforming any continuous spatial query to the corresponding *time-*

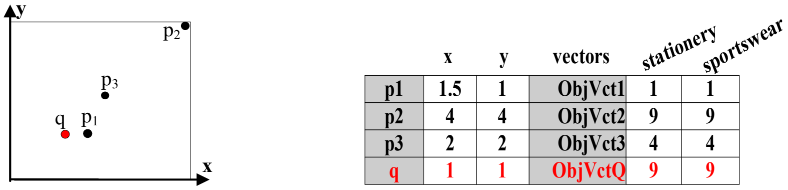
parameterized version query, which returns: (i) the objects that satisfy the corresponding spatial query, (ii) the expiry time of the result, and (iii) the change that causes the expiration of the result.

3.4.5. Cost analysis for reverse k nearest neighbor queries in L_p-norm space. I/O cost analysis for both monochromatic and bichromatic RkNN queries in L_p-norm space are studied in the recent work [Cheema et al. 2011; 2012]. Their methods are based on the Euclidian geometric properties and a concept of influence zone, which is the area such that every point inside this area is the RkNN of query object q and every point outside this area is not the RkNN.

Finally, a preliminary version of this work appears in [Lu et al. 2011]. But in this journal article, we first propose a new cost model to theoretically analyze the performance of algorithms dealing with both location proximity and textual similarities, which is not discussed in any previous literature (to our best knowledge). We also propose a new index tree called C^{2}IUR tree, which considers both location proximity and textual similarity during the construction of the index tree.

4. Baseline Methods

As discussed in Section 3, the similarity metric proposed in Section 2 combines both textual and location information, and therefore the existing methods for RkNN queries cannot be directly employed to handle RSKkNN queries due to the new challenge. One might be tempted to answer an RSKkNN query by separately computing the results for the reverse spatial k nearest neighbors (RSkNN) and the reverse keyword k nearest neighbors (RKkNN), and then select a proper subset from the union of these two results. Besides performance shortcomings, this idea has another serious problem: the result of an RSKkNN query may not even be a subset of the union of the results from the corresponding RSkNN and RKkNN queries. To illustrate, see the example given in Figure 2, assuming k = 1, α = 0.5, we have RSkNN(q) = $\{p_1\}$, RKkNN(q) = $\{p_2\}$, whereas RSKkNN(q) = $\{p_3\}$. This is because q is the nearest neighbor of p_1 by spatial distance and the nearest neighbor of p_2 by textual similarity only. However, by combing spatial and textual distance, q is neither the nearest neighbor of p_1 nor p_2. (In fact, q is the nearest neighbor of p_3.) Therefore, RSKkNN results cannot be directly derived from the union of the results of RSkNN and RKkNN queries.



	x	y	vectors	stationery	sportswear
p1	1.5	1	ObjVct1	1	1
p2	4	4	ObjVct2	9	9
p3	2	2	ObjVct3	4	4
q	1	1	ObjVctQ	9	9

(a) Spatial Distribution

(b) Locations and vectors of points in (a)

Fig. 2. Example for illustrating the relationship of RSkNN, RKkNN and RSKkNN

In the following, we develop two non-trivial baseline algorithms to correctly find *all* answers for RSKkNN queries.

First, for each object $o \in P$, we pre-compute its location proximity and textual similarity, respectively, with each of the other objects to obtain two sorted lists $o.L_s$ and $o.L_t$. In $o.L_s$, the objects are sorted in ascending order of their spatial distance to o, and in $o.L_t$ in descending order of their textual similarity to o. The baseline algorithm is outlined in Algorithm 1. It takes as arguments the database P, the two pre-computed lists for each object in P and the RSKkNN query q. For each object $o \in P$, we find its kth similar object $o.STkNN$ that has the highest score according to the function in Eqn(1). We find $o.stkNN$ by using the threshold algorithm (TA) [Fagin et al. 2003] on the two precomputed lists $o.L_s$ and $o.L_t$ (Line 2). If the spatial-textual similarity between o and its k-th similar object $o.stkNN$ is equal to or larger than the similarity between o and query q, then we prune object o, otherwise, we add o as a result (Line 3-6). Note that this method can also handle dynamic parameters k and α.

For the second baseline, we utilize the available indexes (i.e. IR-trees proposed in [Cong et al. 2009] or [Li et al. 2011]) that combine spatial and textual information to compute the $STkNN$ object of o. That is, for each object o, we find its top-k most similar objects using the existing spatial-textual kNN query techniques, and if the kth result is less than the similarity between o and query q, then o is added to the result set. Therefore, the only difference between the first and the second baseline algorithms lies in Line 2 (Algorithm 1), where the second baseline use the optimized IR-tree to compute the nearest k spatial-textual neighbors. As shown later, we will experimentally compare these two baseline algorithms to each other and also to our proposed algorithms..

ALGORITHM 1: Baseline (P: Database objects, Two pre-computed lists L_s and L_t for each object o in P, q: query)

```
Output: All objects o, s.t. $o \in RSKkNN(q, k, P)$.
1: for each object $o$ in $P$ do
2:     $o.STkNN \leftarrow$ TA($k$, $o.L_s$, $o.L_t$);
3:     if $SimST(o, o.stkNN) \geq SimST(o, q)$ then
4:         Prune object $o$;
5:     else
6:         Report object $o$ as a result;
```

5. A Hybrid Index: IUR-tree

To answer an RSK*k*NN query efficiently, we propose an effective hybrid index called IUR-tree (Intersection-Union R-tree), which is a combination of textual vectors and R-trees [Guttman 1984]. Each node of an IUR-tree contains both spatial location and textual information. Each leaf node contains entries[2] in the form of ($ObjPtr$, $ObjLoc$, $ObjVct$), where *ObjPtr* refers to an object in the database; *ObjLoc* represents the coordinates of the object; and *ObjVct* is the textual vector of the object. A non-leaf node R of IUR-tree contains entries in the form of (Ptr, mbr, $IntUniVct$, cnt), where 1) *Ptr* is the pointer to a child node of R; 2) *mbr* is the MBR of the child node of R; 3) *IntUniVct* is the pointer to two textual vectors: an intersection vector and a union vector. Each item/dimension in a textual vector corresponds to a distinct word that appears in the documents contained in the subtree rooted at Ptr. The weight of each item in the intersection (resp. union) textual vector is the minimum (resp. maximum) weight of the items in the documents contained in the subtree rooted at Ptr. The two vectors are used to compute the similarity approximations (to be presented). Note that the two vectors are not stored inside the nodes of the IUR-tree. The reason is that this guarantees the sizes of all index nodes are the identical and fixed; and 4) *cnt* is the number of objects (in the leaf nodes) in the subtree rooted at Ptr.

Figure 3 illustrates the IUR-tree for the objects in Figure 1. The intersection and union textual vectors are presented in Fig. 5. For example, the weights of item $camera$ in the intersection and union vectors ($IntUniVct2$) of an entry in node $N3$ are 7 and 8, respectively, which are the minimum and maximum weights of the item in the two text vectors $ObjVct7$ and $ObjVct8$ (shown in Fig.1) in node $N1$.

The construction of the IUR-tree is presented in Algorithm 2. It uses an insert operation that is adapted from the corresponding insert operation of the R-tree [Guttman 1984]. To update an IUR-tree incrementally, we use the *order preserving minimal perfect hashing function* (*OPMPHF*)[3] [A.Fox et al. 1991] to organize keywords contained in the subtree of the index node N in the form of $(d_i.p,\, d_i.w), i \in [0, m]$, where m is the total number of words contained in the document of N,

[2] For brevity, objects in the dataset and the index nodes are collectively referred as entries.

[3] The motivation to use this hash function is because of the property of the order preserving. When we lookup a key by reading an OPMPHF-hashed vector, we can stop earlier if the key does not appear in the vector. But other hash functions also work for the purpose of IUR-trees.

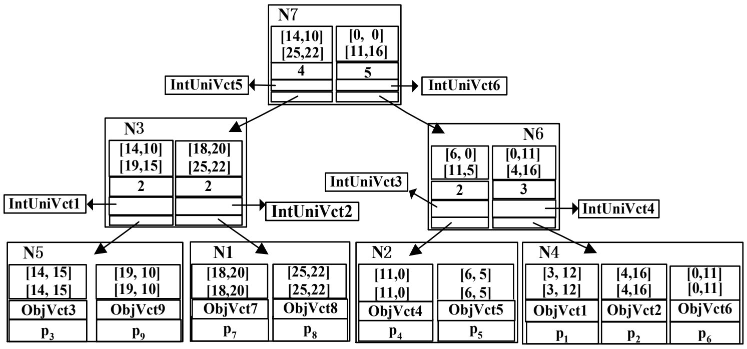
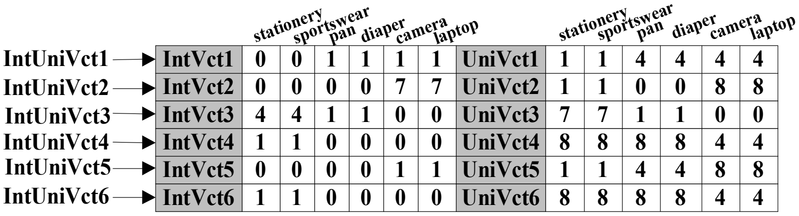


Fig. 3. The IUR-tree of Figure 1



		stationery	sportswear	pan	diaper	camera	laptop		stationery	sportswear	pan	diaper	camera	laptop
IntUniVct1 →	IntVct1	0	0	1	1	1	1	UniVct1	1	1	4	4	4	4
IntUniVct2 →	IntVct2	0	0	0	0	7	7	UniVct2	1	1	0	0	8	8
IntUniVct3 →	IntVct3	4	4	1	1	0	0	UniVct3	7	7	1	1	0	0
IntUniVct4 →	IntVct4	1	1	0	0	0	0	UniVct4	8	8	8	8	4	4
IntUniVct5 →	IntVct5	0	0	0	0	1	1	UniVct5	1	1	4	4	8	8
IntUniVct6 →	IntVct6	1	1	0	0	0	0	UniVct6	8	8	8	8	4	4

Fig. 4. Text vectors for IUR-tree in Figure 2

$d_i.p$ is an integer (position in the word collection) hashed from word d_i using $OPMPHF$, and $d_i.w$ is the weight of word d_i. In particular, in Algorithm 2, Function $Convert()$ in Line 1 is to convert a document to a vector in the form of $(d_i.p, d_i.w)$. Lines 2∽14 use an R-tree based implementation of $ChooseLeaf$ and node-split and -append with text vectors. We modify the standard $AdjustTree$ method to maintain the text description (Lines 15 and 19): if a pair $(d_i.p, d_i.w)$ is inserted to entry E, then the intersection and union vectors of each E's ancestor should be updated recursively.

6. RSKkNN Query Algorithm

In this section, we develop an efficient algorithm to answer RSKkNN queries. At high-level, the algorithm descends the IUR-tree in the branch and bound manner, progressively computing the upper and lower thresholds for each entry E. Then the algorithm decides whether to prune an entry E, to report all objects in E as results, or to consider objects of E as candidates. In the following, we present a novel approach to compute the lower and upper bounds of similarity, denoted $kNN^L(E)$ and $kNN^U(E)$, between a node E in IUR-trees and its kth most similar objects in Section 6.1, and Section 6.2 is dedicated to the details of the algorithm. We summarize the symbols used in this section in Table I.

6.1. Computing Lower and Upper Bounds

For each entry E in an IUR-tree, we need compute the lower and upper bounds of similarity between E and its kth most similar object, denoted by $kNN^L(E)$ and $kNN^U(E)$, respectively.

6.1.1. Similarity Approximations. To efficiently compute $kNN^L(E)$ and $kNN^U(E)$ during IUR-tree traversal, we make full use of each entry traveled by approximating the similarities among entries, and by defining minimal and maximal similarity functions. We first present the definitions for the spatial distance approximation, which is given in previous works (*e.g.*, [N.Roussopoulos et al. 1995; Achtert et al. 2009]), and then concentrate on the new textual part.

ALGORITHM 2: Insert (MBR, $document$)

```
$TextVct$←Convert($document$); //Covert document into text vector in form of $(d_i.p, d_i.w)$.
$N \leftarrow$ ChooseLeaf($MBR$);
add $TextVct$ and $MBR$ to node $N$;
if $N$ needs to be split then
    $\{O, P\} \leftarrow N$.split();
    if $N$.isroot() then
        initialize a new node $M$;
        $M$.append($O.MBR$, $O.TextVct$);
        $M$.append($P.MBR$, $P.TextVct$);
        StoreNode($M$);
        StoreNode($O$);
        StoreNode($P$);
        $R$.RootNode $\leftarrow M$;
    else
        AdjustTree($N.ParentNode$, $O$, $P$);
else
    StoreNode($N$);
    if $\neg N$.isroot() then
        AdjustTree($N.ParentNode$, $N$, null);
```

Table I. Summary of the notations used

E, E'	Two entries (nodes) in IUR-trees
$kNN^L(E)$	The lower bound of similarity between E and its kth most similar objects
$kNN^U(E)$	The upper bound of similarity between E and its kth most similar objects
MinS(E, E')	The minimum spatial distance between the objects in E and E'
MaxS(E, E')	The maximum spatial distance between the objects in E and E'
MinMaxS(E, E')	The minimal overestimation of the spatial distances between the objects in E and E'
MinST(E, E')	The lower bound of spatial-textual similarity between the objects in E and E'
TightMinST(E, E')	A tight lower bound of spatial-textual similarity between the objects in E and E'
MaxST(E, E')	The upper bound of spatial-textual similarity between the objects in E and E'

Spatial distance approximation Given two index entries E and E', we define three distance approximation as follows. i) MinS(E,E') always underestimates the distance between the objects in subtree(E) and subtree(E'): $\forall o \in$ subtree(E), $\forall o' \in$ subtree(E'): dist(o,o')$\geq$ MinS(E,E'); ii) MaxS(E,E') always overestimates the distance between the objects in subtree(E) and subtree(E'): $\forall o \in$ subtree(E), $\forall o' \in$ subtree(E'): dist(o,o') $\leq$ MaxS(E,E'); iii) MinMaxS(E,E') is the minimal distance such that: $\forall o \in$ subtree(E), $\exists o' \in$ subtree(E'): dist(o,o') $\leq$ MinMaxS(E,E'). Intuitively, MinMaxS(E,E') is the minimal overestimation of the distances between the objects in subtree(E) and subtree(E').

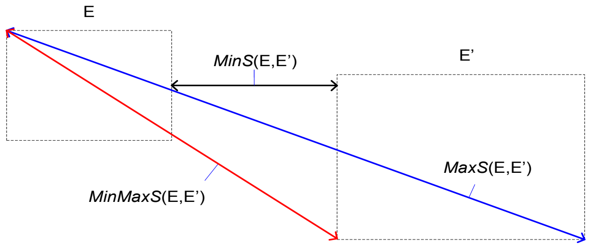


Fig. 5. Illustration of spatial approximation

We assume that the page region of an entry E which is a rectangle is specified by its lower left corner ($E.l_1$, $E.l_2$) and upper right corner ($E.r_1$, $E.r_2$). Furthermore, the center of the page region is denoted by the vector ($E.m_1$, $E.m_2$) with $E.m_i$ = ($E.l_i$ + $E.r_i$)/2. The *MinS*, *MaxS* and *MinMaxS* approximations defined for E and E' can be computed as follows.

MinS(E,E') = $\sqrt{d_1^2 + d_2^2}$, where d_i=$p_i - p'_i$ (i=1 or 2), and

$$p'_i = \begin{cases} p_i = E.r_i, p'_i = E'.l_i & \text{if } E.r_i < E'.l_i \\ p_i = E.l_i, p'_i = E'.r_i & \text{if } E.l_i > E'.r_i \\ d_i = 0 & \text{otherwise} \end{cases} \tag{7}$$

MaxS(E,E') = $\sqrt{d_1^2 + d_2^2}$, where d_i= $p'_i - p_i$ (i=1 or 2), and

$$\begin{cases} p_i = E.l_i, p'_i = E'.r_i & \text{if } E.m_i \leq E'.m_i \\ p_i = E'.l_i, p'_i = E.r_i & \text{otherwise} \end{cases} \tag{8}$$

MinMaxS(E,E') = $\underset{1\leq i\leq 2}{min} \sqrt{(p_i - p'_i)^2 + \underset{1\leq j\leq 2, j\neq i}{max}\{(E.l_j - E'.r_j)^2, (E.r_j - E'.l_j)^2\}}$, where

$$p'_i = \begin{cases} E'.l_i & \text{if } E.m_i{<}E'.m_i \\ E'.r_i & \text{otherwise} \end{cases} \qquad p_i = \begin{cases} E.l_i & \text{if } E.m_i{<}p'_i \\ E'.r_i & \text{otherwise} \end{cases} \tag{9}$$

Note that here we assume that the distance function is L_2-norm. The intuition behind the above formulas is that i) *MinS* is to find the distance between two closest points from two MBRs; ii) *MaxS* is to find the distance between two farthest points from two MBRs; iii) *MinMaxS* is to find the minimal distance such that for any object o in E, we can always find an object o' in E', *MinMaxS* $\geq dist(o,o')$.

Spatial-textual similarity approximation Given two index entries E and E', we define the spatial-textual similarity approximation $MinST(E,E')$, which always underestimates the similarity between the objects in subtree(E) and subtree(E'): $\forall o \in$ subtree(E), $\forall o' \in$ subtree(E'): SimST(o,o') $\geq$ MinST(E,E'). Let the intersection and union textual vectors of an entry E in IUR-tree be $<E.i_1,\cdots,E.i_n>$ and $<E.u_1,\cdots,E.u_n>$, respectively, where n is the total number of words.

DEFINITION 6.1 ($MinST$). *An underestimation of the spatial-textual similarity between two entries E and E' in IUR-tree, denoted by $MinST(E,E')$, is defined as:*

$$MinST(E,E') = \alpha(1 - \frac{MaxS(E,E') - \varphi_s}{\psi_s - \varphi_s}) + (1-\alpha)\frac{MinT(E,E') - \varphi_t}{\psi_t - \varphi_t} \tag{10}$$

where $MaxS(E,E')$ is defined in Equation (8), and

$MinT(E,E')$= $\frac{\sum_{j=1}^{n} E.w_j \times E'.w_j}{\sum_{j=1}^{n} E.w_j^2 + \sum_{j=1}^{n} E'.w_j{}^2 - \sum_{j=1}^{n} E.w_j \times E'.w_j}$,

$$\begin{cases} E.w_j = E.u_j, E'.w_j = E'i_j & \textit{if } E.i_j{*}E.u_j{\geq}E'.i_j{*}E'.u_j \\ E.w_j = E.i_j, E'.w_j = E'.u_j & \textit{otherwise} \end{cases} \tag{11}$$

To understand the above formula about $MinT$, note that the textual similarity between two objects is defined by the extended Jaccard in Equation (4). Informally, given two entries E and E', $MinT$ is derived from the maximum difference of weights for each word (dimension). The formal proof is as follows.

LEMMA 1. *$MinST(E,E')$ satisfies the property that $\forall o \in E$, $\forall$ $o' \in E'$, $SimST(o,o') \geq MinST(E,E')$.*

PROOF. To prove the property of $MinST$ in Lemma 1, we first give a definition called *similarity preserving* function.

DEFINITION 6.2 (SIMILARITY PRESERVING FUNCTION). *Given two functions $fsim$: $V \times V \rightarrow \mathbb{R}$ and $fdim$: $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, where V denotes the domain of n-element vectors and $\mathbb{R}$ the real numbers. We call $fsim$ a similarity preserving function w.r.t. $fdim$, such that for any three vectors $\overrightarrow{p}$=$<x_1, \cdots, x_n>$, $\overrightarrow{p'}$=$< x'_1, \cdots, x'_n>$, $\overrightarrow{p''}$=$<x''_1, \cdots, x''_n>$, if $\forall i \in [1, n]$, $fdim(x_i, x'_i)$ $\geq fdim(x_i, x''_i)$, then we have $fsim(\overrightarrow{p}, \overrightarrow{p'}) \geq fsim(\overrightarrow{p}, \overrightarrow{p''})$.*

CLAIM 1. *Euclidian distance function is a similarity preserving function, w.r.t. function $fdim(x, x') = |x - x'|$.*

Given the Euclidian function $dist(\overrightarrow{X}, \overrightarrow{X'}) = \sqrt{\sum\limits_{i=1}^{n} (x_i - x'_i)^2}$, obviously, we have that if each dimension i, $|x_i - x'_i| \geq |x_i - x''_i|$, then $dist(\overrightarrow{X}, \overrightarrow{X'}) \geq dist(\overrightarrow{X}, \overrightarrow{X''})$. Thus Claim 1 is true.

CLAIM 2. *Extended Jaccard is a similarity preserving function, w.r.t. function $fdim(x, x') = \frac{min\{x,x'\}}{max\{x,x'\}}$, $x, x' > 0$.*

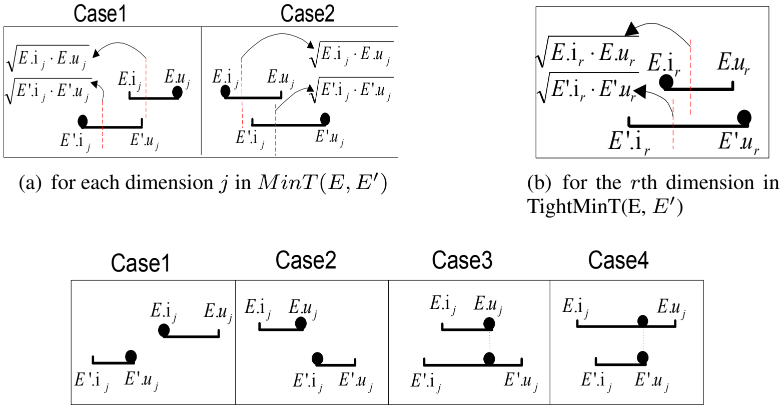
Given x_i, x'_i, x''_i, where $i \in [1, n]$, such that $fdim(x_i, x'_i) \geq fdim(x_i, x''_i)$ and x_i ,x'_i, $x''_i > 0$, we can prove that $\frac{2x_i x'_i}{x_i^2 + x_i'^2} \geq \frac{2x_i x''_i}{x_i^2 + x_i''^2}$, then we can derive inequality (12) is true by means of mathematical induction. Thus extended Jaccard is a similarity preserving function, *i.e.*, if $\forall i \in [1, n]$, $x_i \leq \sqrt{x'_i x''_i}$, and $x'_i \leq x''_i$, then $EJ(\vec{p}, \vec{p'}) \geq EJ(\vec{p}, \vec{p''})$. Therefore Claim 2 holds.

$$\begin{aligned}
&\frac{\sum_{i=1}^{n} x_i x'_i}{\sum_{i=1}^{n} \frac{x_i^2 + x_i'^2}{2}} \geq \frac{\sum_{i=1}^{n} x_i x''_i}{\sum_{i=1}^{n} \frac{x_i^2 + x_i''^2}{2}} \qquad (12)\\
\Longrightarrow\ &\frac{\sum\limits_{i=1}^{n} x_i x'_i}{\sum\limits_{i=1}^{n} x_i^2 + \sum\limits_{i=1}^{n} x_i'^2 - \sum\limits_{i=1}^{n} x_i x'_i} \geq \frac{\sum\limits_{i=1}^{n} x_i x''_i}{\sum\limits_{i=1}^{n} x_i^2 + \sum\limits_{i=1}^{n} x_i''^2 - \sum\limits_{i=1}^{n} x_i x''_i}\\
\Longrightarrow\ &EJ(\overrightarrow{p}, \overrightarrow{p'}) \geq EJ(\overrightarrow{p}, \overrightarrow{p''})
\end{aligned}$$

Based on Claim 2, we proceed to prove the property of $MinST$, which is the fusion of $MaxS$ and $MinT$.

$MinT$ in Eqn(11): For each dimension j, as shown in Figure 6(a), when $\sqrt{E.i_j \cdot E.u_j} \geq \sqrt{E'.i_j \cdot E'.u_j}$ (Case 1), *i.e.*, $\frac{E'.i_j}{E.u_j} \leq \frac{E.i_j}{E'.u_j}$, then for $\forall E.w \in [E.i_j, E.u_j]$ and $\forall E'.w \in [E'.i_j, E'.u_j]$, we have $\frac{E'.i_j}{E.u_j} \leq \frac{min\{E.w_j, E'.w_j\}}{max\{E.w_j, E'.w_j\}}$. Thus according to Claim 2, the assignments $E.w_j = E.u_j$, $E'.w_j = E'.i_j$ can guarantee that $MinST(E, E')$ is the minimum similarity between two entries E and E', *i.e.*, $\forall$ $o \in subtree(E)$, $\forall o' \in subtree(E')$, $MinT(E, E') \leq SimT(o, o')$. And for Case 2, the property of $MinST$ can be similarly proved.

For $MinST$ in Eqn(10), since $\forall o \in E$, $\forall o' \in E'$ are enclosed in the MBRs of index nodes E and E' respectively, the maximum Euclidian distance between E and E' $MaxS(E, E')$ is no less than the Euclidian distance between o and o', *i.e.*, $MaxS(E, E') \geq dist(o, o')$, thus $\alpha(1 - \frac{MaxS(E,E') - \varphi_s}{\psi_s - \varphi_s})$ $\leq \alpha(1 - \frac{dist(o,o') - \varphi_s}{\psi_s - \varphi_s})$, where φ_s, ψ_s are constants and $\alpha \in$[0, 1]. And as proved above that $\forall o \in E$, $\forall o' \in E'$, $MinT(E, E') \leq SimT(o, o')$. Thus Eqn(10) can guarantee that $\forall o \in E$, $\forall o' \in E'$, $MinST(E, E') \leq SimST(o, o')$, which concludes the proof. □


Case1
Case2
$\sqrt{E.i_j \cdot E.u_j}$
$\sqrt{E'.i_j \cdot E'.u_j}$
$E.i_j$
$E.u_j$
$E'.i_j$
$E'.u_j$
$\sqrt{E.i_r \cdot E.u_r}$
$\sqrt{E'.i_r \cdot E'.u_r}$
$E.i_r$
$E.u_r$
$E'.i_r$
$E'.u_r$
Case1
Case2
Case3
Case4

(a) for each dimension j in $MinT(E, E')$

(b) for the rth dimension in TightMinT(E, E')

(c) for each dimension j in $MaxT(E, E')$

Fig. 6. Illustration to the estimation of *MinT*, *TightMinT* and *MaxT*

Lemma 1 suggests that there are at least $|E'|$ objects o' in E' s.t. $\forall o \in E$, $SimST(o, o') \geq MinST(E, E')$. Therefore, we can use $MinST(E, E')$ to estimate the lower bound $kNN^L(E)$ that should be greater than $MinST(E, E')$.

We next propose another similarity definition which is larger than $MinST(E, E')$ and thus may be used as a tighter bound estimation.

DEFINITION 6.3 ($TightMinST$). *A tight lower bound of spatial-textual similarity between two entries E and E' in IUR-tree, denoted as $TightMinST(E, E')$, is defined as:*

$$TightMinST(E, E') = max\left\{ \begin{array}{l} \alpha(1 - \dfrac{MinMaxS(E, E') - \varphi_s}{\psi_s - \varphi_s}) + (1 - \alpha)\dfrac{MinT(E, E') - \varphi_t}{\psi_t - \varphi_t}, \\ \alpha(1 - \dfrac{MaxS(E, E') - \varphi_s}{\psi_s - \varphi_s}) + (1 - \alpha)\dfrac{TightMinT(E, E') - \varphi_t}{\psi_t - \varphi_t} \end{array} \right\} \quad (13)$$

where, $MinMaxS(E, E')$ [Achtert et al. 2009] is showed in Equation (9).

$$TightMinT(E, E') = \max_{1 \leq r \leq n} \frac{E.w_r \times E'.w_r + \sum_{j=1, j \neq r}^{n} E.w_j \times E'.w_j}{E.w_r^2 + E'.w_r^2 - E.w_r \times E'.w_r + \sum_{j=1, j \neq r}^{n} (E.w_j^2 + E'.w_j^2 - E.w_j \times E'.w_j)} \quad (14)$$

$$E'.w_r = \begin{cases} E'.u_r & \text{if } E.i_r * E.u_r > E'.i_r * E'.u_r \\ E'.i_r & \text{otherwise} \end{cases} \quad (15)$$

$$E.w_r = \begin{cases} E.i_r & \text{if } \sqrt{E.i_r * E.u_r} < E'.w_r; \\ E.u_r & \text{otherwise}; \end{cases} \quad (16)$$

and $E.w_j$ and $E'.w_j$ are assigned as Eqn(11).

Intuitively, the particular reason why $TightMinST$ can provide a tighter lower bound than $MinST$ is that $TightMinST$ guarantees that there is at least one object $o' \in E'$ s.t. $\forall o \in E$, $SimST(o, o') \geq TightMinST(E, E')$. But $MinST$ can guarantee that $\forall o \in E$, $\forall o' \in E'$,

$SimST(o,o') \geq MinST(E,E')$. Therefore, this different property gives us an extra opportunity to carve out a tighter bound. The formal description and proof are as followed.

LEMMA 2. *$TightMinST(E,E')$ has the property that $\exists\ o' \in E'$ s.t. $\forall o \in E$, $SimST(o,o') \geq TightMinST(E,E')$.*

PROOF. Eqn(13) suggests that $TightMinST$ is composed of $MinMaxS$, $MinT$, $MaxS$ and $TightMinT$, which have the following properties respectively.

$MinMaxS$ satisfies that $\exists o' \in E'$, s.t. $\forall o \in E, ,\ dist(o,o') \leq MinMaxS(E,E')$.

$MinT$ satisfies that $\forall o \in E, \forall o' \in E, EJ(o,o') \geq MinT(E,E')$.

$TightMinT$ in Eqn(14): As shown the assignment of one dimension r in Figure 6(b), when $\sqrt{E'.i_r \cdot E'.u_r} < \sqrt{E.i_r \cdot E.u_r}$, let $E'.w_r = E'.u_r$, then $\exists E'.w_r \in\ [E'.i_r, E'.u_r]$, $\frac{min\{E.w_r, E'.w_r\}}{max\{E.w_r, E'.w_r\}} \leq \frac{min\{E.w_r, E'.u_r\}}{max\{E.w_r, E'.u_r\}}$. Then given $E'.w_r > \sqrt{E.i_r \cdot E.u_r}$, let $E.w_r = E.i_r$ so that $\forall\ E.w_r \in [E.i_r, E.u_r]$, $\frac{min\{E.w_r, E'.w_r\}}{max\{E.w_r, E'.w_r\}} \geq \frac{min\{E.i_r, E'.w_r\}}{max\{E.i_r, E'.w_r\}}$, Additionally, the rest dimension weights $E.w_j$ and $E'.w_j$ are assigned as Figure 6(a). Therefore, according to Claim 2, there exists an object $o' \in E'$, the rth dimension of which is $E'.u_r$, so that $\forall o \in E$, $SimT(o,o') \geq TightMinT(E,E')$. Finally, to make the approximation accurate, we take the maximum as the final approximation for $TightMinT$.

$TightMinST(E,E')$ in Eqn(13): Since $\exists\ o' \in E'$, $\forall\ o \in E$, $dist(o,o') \leq MinMaxS(E,E')$, moreover, since $\forall o'' \in E$, $\forall o \in E$, $EJ(o,o'') \geq MinT(E,E')$, so for $o' \in E'$, it is also true that $EJ(o,o') \geq MinT(E,E')$. Thus $\exists o' \in E'$, $\forall o \in E$, $SimST(o,o') = \alpha(1 - \frac{dist(o,o') - \varphi_s}{\psi_s - \varphi_s}) + (1 - \alpha)\frac{EJ(o,o') - \varphi_t}{\psi_t - \varphi_t} \geq \alpha(1 - \frac{MinMaxS(E,E') - \varphi_s}{\psi_s - \varphi_s}) + (1 - \alpha)\frac{MinT(E,E') - \varphi_t}{\psi_t - \varphi_t}$. Similarly, $\exists o' \in E'$, $\forall o \in E$ $SimST(o,o') \geq \alpha(1 - \frac{MaxS(E,E') - \varphi_s}{\psi_s - \varphi_s}) + (1 - \alpha)\frac{TightMinT(E,E') - \varphi_t}{\psi_t - \varphi_t}$. To make the approximation accurate, the final approximation of $TightMinST(E,E')$ is the maximum one with the guarantee of satisfying the corresponding property. □

As suggested from Lemma 2, there is at least one object o' in E' s.t. $\forall o \in E$, $SimST(o,o') \geq TightMinST(E,E')$. Hence, unlike $MinST$ which can contribute $|E|$ objects, $TightMinST$ can contribute only one object to be the kNNs of E', but $TightMinST$ is much tighter than $MinST$.

DEFINITION 6.4 ($MaxST$). *An overestimation of the spatial-textual similarity between two entries E and E' in IUR-tree, denoted as $MaxST(E,E')$, is defined as:*

$$MaxST(E,E') = \alpha(1 - \frac{MinS(E,E') - \varphi_s}{\psi_s - \varphi_s}) + (1 - \alpha)\frac{MaxT(E,E') - \varphi_t}{\psi_t - \varphi_t} \tag{17}$$

where $MinS(E,E')$ is defined in Equation (7); and $MaxT(E,E')$ is:

$$\frac{\sum_{j=1}^{n} E.w_j \times E'.w_j}{\sum_{j=1}^{n} E.w_j^2 + \sum_{j=1}^{n} E'.w_j{}^2 - \sum_{j=1}^{n} E.w_j \times E'.w_j}$$

$$\begin{cases} E.w_j = E.i_j,\ E'.w_j = E'.u_j & \text{if } E.i_j > E'.u_j \\ E.w_j = E.u_j,\ E'.w_j = E'.i_j & \text{if } E.u_j < E'.i_j \\ E.w_j = E'.w_j = E.u_j & \text{if } E'.i_j \leq E.u_j \leq E'.u_j \\ E.w_j = E'.w_j = E'.u_j & \text{otherwise.} \end{cases} \tag{18}$$

LEMMA 3. *$MaxST(E,E')$ has the property that $\forall\ o' \in E'$, $\forall o \in E$, $SimST(o,o') \leq MaxST(E,E')$.*

The proof is similar to that in Lemma 1 and omitted here.

COROLLARY 6.5. *There is at most one object o' in E' s.t. $\forall o \in E$, $SimST(o, o') \leq MaxST(E, E')$.*

Note that Lemma 1 $\sim$ 3 also hold when the two entries E and E' in IUR-tree are identical, i.e., E = E'.

6.1.2. Lower and Upper Bound Contribution Lists. We are ready to explore the similarity approximations defined above to identify the lower and upper bounds $kNN^L(E)$, $kNN^U(E)$ of the most similar k objects for each entry E.

DEFINITION 6.6 (LOWER BOUND CONTRIBUTION LIST).
Let $\mathcal{T}$ be a set of entries in IUR-trees that do not have ancestor-descendant relationships. Given an entry $E \in \mathcal{T}$, a lower bound contribution list of E, denoted as $E.^LCL$, is a sequence of t (1≤t≤k) triples $<s_i, E'_i, num_i>$ sorted in descending order of s_i, where $E' \in \mathcal{T}$, s_i is $MinST(E, E')$ or $TightMinST(E, E')$, and

$$num_i = \begin{cases} |E'| - 1 & \textit{if } s_i{=}MinST(E, E') \textit{ and } E' {\neq} E \\ |E'| - 2 & \textit{if } s_i{=}MinST(E, E') \textit{ and } E'{=}E \\ 1 & \textit{otherwise.} \end{cases}$$

such that t is the minimal number fulfilling $\sum_{i=1}^{t} num_i \geq k$.

For s_i = $MinST(E, E')$ either when $E' {\neq} E$ or $E'{=}E$, the rationale for subtracting one from $|E'|$ ($|E'|$-1 when $E'{=}E$) is due to the potential presence of one object in E' with more precise approximation by $TightMinST$.

Example 6.7. Given k=3, three entries E, E'_1, E'_2 in IUR-tree. Suppose the number of objects in E, denoted as $|E|$, is 3, and $|E'_1|$=2, $|E'_2|$=3. Furthermore,

$$\begin{aligned} &MinST(E, E) = 0.85, num = 1; \quad TightMinST(E, E) = 0.85, num = 1 \\ &MinST(E, E'_1) = 0.55, num = 1; \quad TightMinST(E, E'_1) = 0.61, num = 1 \\ &MinST(E, E'_2) = 0.72, num = 2; \quad TightMinST(E, E'_2) = 0.82, num = 1 \end{aligned}$$

Then we sort the similarity approximations above in descending order obtaining <0.85, 0.85, 0.82, 0.61, 0.55>. Since $\sum_{i=1}^{3} num_i = 1 + 1 + 2 \geq k = 3$, thus we get the lower bound contribution list of E is $\Big\langle <0.85, E, 1>, <0.85, E, 1>, <0.82, E'_2, 2> \Big\rangle$.

Following the notations in Definition 6.6, we have the following lemma.

LEMMA 4. *If the t-th element (i.e., $E.^LCL.s_t$) is larger than or equal to the maximal similarity between E and q (denoted by $MaxST(E, q)$), no answer exists in subtree(E), the subtree rooted at E, and thus we can safely prune subtree(E).*

PROOF. The definition 6.6 for the lower bound contribution list of entry E is composed of $MinST(E, E')$ and $TightMinST(E, E')$. If $E' {\neq} E$, then there are at least one object o' in E' such that $\forall o \in E, TightMinST(o, o') \geq MinST(E, E')$, and there are $|E'| - 1$ objects o'' ($o'' \neq o'$) such that $\forall o \in E, SimST(o, o') \geq MinST(E, E')$. While, if $E' {\neq} E$, then for each object $o \in E$, there are at least one object o' ($o' \neq o$) in E' such that $SimST(o, o') \geq TightMinST(E, E')$, and there are $|E'| - 2$ objects o'' ($o'' \neq o' \&\& o'' \neq o$) such that $SimST(o, o') \geq MinST(E, E')$.

Thus $E.^LCL.s_t$ obtained from Definition 6.6 satisfies that for all the objects $o {\in} E$, there are at least k objects o' ($o' {\neq} o$) such that $SimST(o, o') \geq E.^LCL.s_t$. If the condition in Lemma 4 holds, i.e., $E.^LCL.s_t \geq MaxST(E, q)$, then for all the objects $o {\in} E$, there are at least k distinct objects o' ($o' {\neq} o$) such that $SimST(o, o') \geq SimST(o, q)$. Hence we can safely prune away $subtree(E)$, avoiding the traverse of $subtree(E)$ during query processing. □

Thus we let the lower bound $kNN^L(E)$ be $E.^LCL.s_t$. That is we can prune E if $kNN^L(E) \geq MaxST(E, q)$.

DEFINITION 6.8 (UPPER BOUND CONTRIBUTION LIST).
Let $\mathcal{T}$ be a set of entries in the IUR-tree that do not have ancestor-descendant relationships. Given an entry $E \in \mathcal{T}$, an upper bound contribution list of E, denoted as $E.^UCL$, is a sequence of t ($1 \leq t \leq k$) triples $<s_i, E'_i, num_i>$ sorted in descending order of s_i, where $E' \in \mathcal{T}$, s_i is $MaxST(E, E')$ and

$$num_i = \begin{cases} |E'| & \text{if } E' \neq E \\ |E'| - 1 & \text{otherwise.} \end{cases}$$

such that t is the minimal number fulfilling $\sum\limits_{i=1}^{t} num_i \geq k$.

Example 6.9. *Given k=3, three entries E, E'_1, E'_2 in IUR-tree. Suppose the objects number in E, denoted as $|E|$, is 3, and $|E'_1|$=2, $|E'_2|$=3. Furthermore,*

$$MaxST(E, E) = 1, \ \ num = 2;$$
$$MinST(E, E'_1) = 0.66, \ \ num = 2;$$
$$MinST(E, E'_2) = 0.88, \ \ num = 3$$

Then we sort the similarity approximations above in descending order obtaining <1, 0.88, 0.66>. Since $\sum_{i=1}^{2} num_i = 2 + 2 \geq k = 3$, thus we get the upper bound contribution list of E is $\big\langle < 1, E, 2 >, < 0.88, E'_2, 3 > \big\rangle$.

Following the notations in Definition 6.8, we have the following lemma.

LEMMA 5. *If the t-th element (i.e., $E.^UCL.s_t$) is smaller than the minimal similarity between E and q (denoted by $MinST(E, q)$), then q must be one of the top-k most similar objects for all objects in E, and objects in E are included as results.*

PROOF. The definition 6.6 for the upper bound contribution list of entry E is composed of $MaxST(E, E')$. If $E' \neq E$, then for all the objects $o \in E$, there are $|E'|$ objects $o' \in E'$ ($o' \neq o$) such that $SimST(o, o') \leq MaxST(E, E')$, i.e., there is at most one object $o' \in E'$ ($o' \neq o$) such that $SimST(o, o') \geq MaxST(E, E')$. While if E' = E, then for all the objects $o \in E$, there are $|E'| - 1$ objects $o'' \in E'$ such that $SimST(o, o'') \leq MaxST(E, E')$.

Thus $E.^UCL.s_t$ obtained from Definition 6.8 satisfies that for all the objects $o \in E$, there are at most k objects o' ($o' \neq o$) such that $SimST(o, o') \geq E.^UCL.s_t$. If the condition in Lemma 5 holds, i.e., $E.^UCL.s_t \leq MinST(E, q)$, then for all the objects $o \in E$, there are at most k distinct objects o' ($o' \neq o$) such that $SimST(o, o') \leq SimST(o, q)$. Thus all objects in E will be reported as part of the answer. □

Note that the upper bound $kNN^U(E)$ is exactly $E.^UCL.s_t$. That is, as shown in Lemma 5, we can report E to be a result entry if $kNN^U(E) < MinST(E, q)$. Intuitively, this is because $kNN^U(E)$ is the smallest similarity for objects to be one of kNNs of E. Since MinST(E, q) is greater than $kNN^U(E)$, q is the kNN object of E. In other words, all objects in E are RSKkNN of q.

Figure 7 illustrates the strategies of using $kNN^L(E)$ and $kNN^U(E)$ to determine whether the entry E is a result. The similarity approximations in $E.^LCL$ (resp. $E.^UCL$) are within the shaded ring "L" (resp. "U"). Specially, the dashed line in "L" (resp. "U") is $kNN^L(E)$ (resp. $kNN^U(E)$). Note that the circle that is farther away from E indicates the similarity between object on the circle and entry E is smaller. If $q3$ is the query object, we can prune E since the similarity $MaxST(E, q)$ between E and $q3$ is within the dashed ring $kNN^L(E)$ (*i.e.*, equal to or larger than $kNN^L(E)$). If $q1$ is the query, we report E as a result entry since the similarity $MinST(E, q)$ between E and $q1$

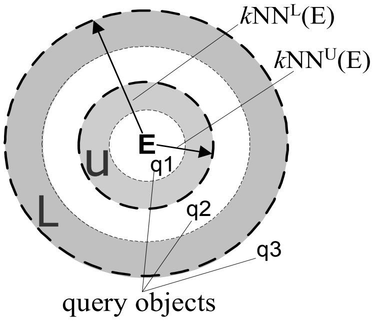


Fig. 7. Illustration of pruning and reporting results using $kNN^L(E)$ and $kNN^U(E)$

is within the ring $kNN^U(E)$. If the query is $q2$, we cannot determine whether E belongs to results based on $kNN^L(E)$ and $kNN^U(E)$.

Extension to Cosine Similarities. Recall that besides the extended Jaccard, the textual similarity can also be measured by other models such as the cosine similarity (see Equation 5). In order to adapt our algorithm for the cosine similarity, we only need to change the textual similarity approximations of $MinT$ in Eqn(11) and $MaxT$ in Eqn. (18). In particular, the minimal cosine textual similarity $MinT_{cos}(E, E')$ between two entries E and E' is given in Eqn(19) and the maximal cosine textual similarity $MaxT_{cos}(E, E')$ is defined to be 1. It is not hard to prove $\forall o' \in E', \forall o \in E, Cosine(o, o') \geq MinT_{cos}(E, E')$, and $\forall o' \in E', \forall o \in E, Cosine(o, o') \leq MaxT_{cos}(E, E')$. Hence Lemmas 1 and 3 still hold under the cosine similarity.

$$MinT_{cos}(E, E') = \frac{\sum_{j=1}^{n} E.i_j \times E'.i_j}{\sqrt{\sum_{j=1}^{n} E.u_j^2} * \sqrt{\sum_{j=1}^{n} E'.u_j{}^2}} \tag{19}$$

6.2. Search Algorithm

We proceed to develop an efficient algorithm to answer RSKkNN queries (see Algorithm 3 and 4). At high-level, the algorithm descends the IUR-tree in the branch and bound manner, progressively computing the thresholds $kNN^L(E)$ and $kNN^U(E)$ for each entry E by inheriting and updating the lower and upper contribution lists. Based on the thresholds, the algorithm then decides whether to prune an entry E, to report all objects in E as results, or to consider child entries of E as candidates.

The algorithm uses the following data structures: a max-priority queue U, which stores nodes E associated with the priority $MaxST(E, q)$, a candidate object list COL that needs to be checked, a pruned entry list PEL, and a result object list ROL.

The algorithm begins with initialization and then enqueues the root of the IUR-tree into U (Line 1–2 in Algorithm 3). When U is not empty (Line 3 in Algorithm 3), we dequeue the entry P from U with the highest priority (Line 4 in Algorithm 3). For each child entry E of P, E first *inherits* the upper/lower bound lists of P (which is discussed in more details later)(Line 6 in Algorithm 3), based on which, we determine whether E is a result entry ("*hit*") or can be pruned ("*drop*") by invoking procedure IsHitOrDrop (Line 7 in Algorithm 3). If E can be pruned, E is added to PEL (Line 10-11 in Algorithm 4), and if E is reported as a result entry , E is added to ROL (Line 14-15 in Algorithm 4); Otherwise, we use E to "*mutually effect*" $E' \in COL \cup ROL \cup U$ to update the upper/lower bound contribution lists to mutually tighten their upper/lower bounds (Line 9 in Algorithm 3). Note that entries E' are selected in decreasing order of $MaxST(E, E')$ since entries E' with higher $MaxST(E, E')$ are more likely to be within the kNN of E (Line 8 in Algorithm 3). If E' is pruned or reported as a result entry then remove E' from its original data structure U or COL (Line 13–14 in Algorithm 3). If E is determined as a hit or drop, then consider next child entry of P (Line 10 in Algorithm 3). If E still cannot be determined whether to be a result entry after effected

```
ALGORITHM 3: RSK$k$NN ($R$: IUR-tree root, $q$: query)
Output: All objects $o$, s.t. $o \in RSKkNN(q, k, R)$.
 1: Initialize a priority queue $U$, and lists $COL$, $ROL$, $PEL$;
 2: EnQueue($U$, $R$);
 3: while $U$ is not empty do
 4:     $P \leftarrow$ DeQueue($U$);//Priority of $U$ is $MaxST(P, q)$
 5:     for each child entry $E$ of $P$ do
 6:         Inherit($E.CLs$, $P.CLs$);
 7:         if (IsHitOrDrop($E$, $q$)=false) then
 8:             for each entry $E'$ in $COL$, $ROL$, $U$ in decreasing order of $MaxST(E, E')$ do
 9:                 UpdateCL($E$,$E'$);//update contribution lists of $E$.
10:                 if (IsHitOrDrop($E$, $q$)=true) then break;
11:                 if $E' \in U \cup COL$ then
12:                     UpdateCL($E'$,$E$);//update contribution lists of $E'$ using $E$.
13:                     if (IsHitOrDrop($E'$, $q$)=true) then
14:                         Remove $E'$ from $U$ or $COL$;
15:             if ($E$ is not a hit or drop) then
16:                 if $E$ is an index node then
17:                     EnQueue($U$, $E$);
18:                 else $COL$.append($E$); //a database object
19: FinalVerification($COL$, $PEL$, $ROL$);
```

by all the entries in COL, ROL and U, then add E to the corresponding list or queue (Line 15–18 in Algorithm 3). Finally, when the priority queue U is empty, we still need to process objects in the candidate list COL to decide if they are part of answers by invoking Procedure FinalVerification (Line 19 in Algorithm 3).

Note that here we adopt a tricky idea called "*lazy travel-down*" for each entry E' in the pruned list PEL to save I/O cost. That is, in Line 8 of Algorithm 3, we do not access the subtree of $\forall E' \in PEL$ to affect entry E that is processed currently until we reach the final verification phase. In this way, as shown in the experimental section, "*lazy travel-down*" accelerates the query processing by avoiding the scan of many portions of the IUR-tree.

Procedure FinalVerification in Algorithm 4: it is to determine if the candidate objects in COL are "*hits*" or "*drops*". The main idea is to check the effect of the entries in PEL on each candidate in COL. Specifically, we update the contribution lists for candidates in COL until we can correctly determine if each candidate object belongs to an answer or not. In particular, Line 2 selects the entry E in PEL which has the lowest level in the IUR-tree. This is because the entries in the lower level often have the tighter bounds than those in the higher level and thus they are more likely to identify whether the candidates are results. Line 4 uses the entry E to update the contribution list of each candidate o in COL and Line 5 checks if o can be removed from the candidate list. Finally, we add children of E into PEL since they may also affect the candidates in COL (Line 8–9). This process continues until COL becomes empty.

In particular, Line 6 in Algorithm 3 introduces an efficient technology called **Inherit**, *i.e.*, a child entry inherits (copies) the contribution lists from its parent entry. Inherit makes use of the parent nodes to avoid computing contribution lists from the scratch, and thus reducing runtime (to be shown in our experimental results). However, inherit will lead to a problem called *object conflict*: the same object in the contribution lists of a child entry may be counted twice (one from the inheritance of parent entry and the other one from itself after other entries' affecting), resulting in wrong upper or lower bounds of the child entry. In order to avoid such a problem, Line 18–20 in Algorithm 4 guarantee that there is no object in contribution lists which is double counted, as illustrated in the following example.

ALGORITHM 4: Procedures in **RSKkNN**

```
Procedure FinalVerification($COL$, $PEL$, $q$)
while ($COL \neq \emptyset$) do
    Let $E$ be an entry in $PEL$ with the lowest level;
    $PEL$=$PEL$-$\{E\}$;
    for each object $o$ in $COL$ in decreasing order of $MaxST(E,o)$ do
        UpdateCL($o$,$E$);//update contribution lists of $o$.
        if (IsHitOrDrop($o$, $q$)=true) then
            $COL$=$COL$-$\{o\}$;
    for each child entry $E'$ of $E$ do
        $PEL$=$PEL \cup \{E'\}$; //access the children of $E'$
Procedure IsHitOrDrop($E$: entry, $q$: query)
if $kNN^L(E) \geq MaxST(E,q)$ then
    $PEL$.append($E$); //Lemma 4
    return true;
else
    if $kNN^U(E) < MinST(E,q)$ and $E$ is the rightest child entry then
        $ROL$.append($subtree(E)$); //Lemma 5
        return true;
    else return false;
Procedure UpdateCL($E$: entry, $E'$: entry)
for each tuple $<s_i,E'_i,num_i> \in E.^LCL$ do
    if $E'_i$=$E$ or $E'_i$=Parent($E$) then
        remove $<s_i,E'_i,num_i>$ from $E.^LCL$; //Clean Conflicts
if $kNN^U(E) < MaxST(E,E')$ then
    $E.^UCL \leftarrow$Top$k$Max($E.^UCL$, $MaxST(E,E')$, 1);
if $kNN^L(E) < TightMinST(E,E')$ then
    $E.^LCL \leftarrow$Top$k$Max($E.^LCL$, $TightMinST(E,E')$, 1);
if $kNN^L(E) < MinST(E,E')$ then
    $E.^LCL \leftarrow$Top$k$Max($E.^LCL$, $MinST(E,E')$, $|E'|$-1);
SubProcedure TopkMax($L$,$f(E,E')$,$C$)
Return the $t$-th triple in contribution list $L$, where $t$ is the minimal number fulfilling $\sum_{i=1}^{t} L.num_i \geq k$.
```

Table II. Trace of RSKkNN algorithm in Example1

Steps	Actions	U	COL	ROL	PEL
1	Dequeue N7; Enqueue N3,N6	N6,N3	∅	∅	∅
2	Dequeue N6; Enqueue N2, N4	N2, N3, N4	∅	∅	∅
3	Dequeue N2;	N3, N4	p5	p4	∅
4	Dequeue N3; Enqueue N5	N5, N4	p5	p4	N1
5	Dequeue N5	N4	p5, p9	p4	N1, p3
6	Dequeue N4	∅	p9	p4, p1, p5	N1, p2, p3, p6
7	Verify p9	∅	∅	p4, p1, p5, p9	N1, p2, p3, p6

Example 6.10. We use this example to illustrate RSKkNN algorithm. Consider the dataset in Figure 1 and a query object $q(12,6)$, $q.vct$ $=$ $<(stationary, 8), (sportswear, 8)>$, and let k=2, α=0.6. The algorithm starts by enqueueing N7 into a priority queue U, and the trace of the algorithm is shown in Table 1. The query answers have four objects: $p1$, $p4$, $p5$ and $p9$, as shown in Step 7 of Table 1.

Here we focus on Step 4 of Table 1 to illustrate the *mutual-effect* strategy and *inherit* technology (See Figure 4). After $N3$ is dequeued from U in Step 4, we access its child entries $N5$ and $N1$.

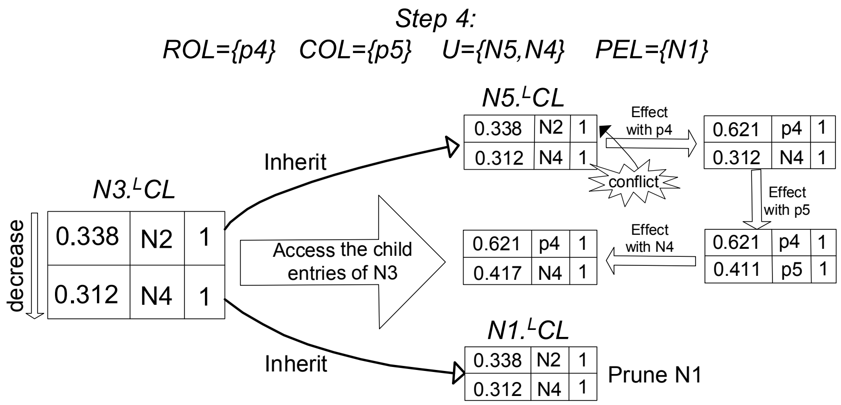


Fig. 8. Illustration to RSKkNN algorithm

1) $N5$ also inherits the contribution list from $N3$ (Line 6 in Algorithm 3). However, it can neither be pruned nor be determined to be results (Line 7 in Algorithm 3). Thus it will "mutual-effect" with $p4$, $p5$, and $N4$ (Line 8–14 in Algorithm 3). When we consider the effect of $p4$ on the contribution list ($N5.^{L}CL$) of $N5$ (Line 9 in Algorithm 3), $N2$ (inherited from $N3.^{L}CL$) in $N5.^{L}CL$ conflicts with $p4$ since $p4$ is a child of $N2$ and $N2$ may contribute the same object $p4$ for $N5.^{L}CL$. To solve the conflict, we remove $N2$ from $N5.^{L}CL$ (Line 20 in Algorithm 4), and add $p4$ to $N5.^{L}CL$ with a more accurate estimation (0.621) of similarity with $N5$ than $N2$. The triple <0.621, $p4$, 1> is added in $N5.^{L}CL$. Next in a similar way, we use $p5$ and $N4$ to effect with $N5$, respectively. Finally $N5$ still cannot be determined to be a hit or drop and it is enqueued to U (Line 17 in Algorithm 3).

2) $N1$ inherits the contribution list from $N3$ (Line 6 in Algorithm 3) and is pruned immediately according to Lemma 4 (Line 11 in Algorithm 4) without having effect on any other entries, which illustrates the benefit of inherit technology. This is because $MaxST(N1, q)$= 0.308 is smaller than $N3.^{L}CL.s_2$=0.312, thus $MaxST(N1, q) \leq TightMinST(N1,\ N4) \leq TightMinST(N1,\ N2)$, *i.e.*, there are at least two objects o' in $N4$ and $N2$, s.t. $\forall o \in N1$, $SimST(o, o') \geq TightMinST(N1,q)$, therefore we can prune $N1$ according to Lemma 4.

Theorem 1. *Given an integer k, a query q and an index tree R, Algorithm 3 correctly returns all RSKkNN points.*

PROOF. We prove that (1) Algorithm RSKkNN does not return false positive, that is all returned objects are the desired answers; and that (2) the returned results are complete (no false negative).

Correctness: The search strategy in RSKkNN algorithm is to prune entries E in the tree using the lower bound of spatial-textual kNN of E: $kNN^{L}(E)$ (Line 10-11 in Algorithm 4) and to report entries E using the upper bound $kNN^{U}(E)$ (Line 14-15 in Algorithm 4). $kNN^{L}(E)$ is calculated by means of $MinST$ and $TightMinST$ in the lower-bound contribution list of E. According to the properties of $MinST$ and $TightMinST$ in Lemma 1 and Lemma 2, entry E can be safely pruned if $MaxST(E, q) \leq kNN^{L}(E)$ since it can guarantee that there are at least k objects whose similarities are larger than or equal to the maximum similarity between E and query object q. Analogously, based on Lemma 3, entry E can be safely reported as a result entry if $MinST(E, q) > kNN^{U}(E)$ with the condition that there are at most k objects (among all the objects) whose similarities are smaller than $MinST(E, q)$.

Furthermore, based on the observation that the similarity approximations of ancestor entries are more conservative than that of descendant entries, we can prove the correctness of the techniques of "inherit" (Line 6 in Algorithm 3) and "lazy travel-down" (Line 8 in Algorithm 3), respectively.

CLAIM 3. *"Inherit" technology used in Algorithm 3 is correct.*

PROOF: We need to prove that an entry in IUR-tree can be pruned or be reported as results safely using the contribution lists "inherited" from its parent entry. That is, given a child entry C and the lower (resp. upper) bound contribution list $P.^{L}CL$ (resp. $P.^{U}CL$) of its parent entry P, we can prune entry C if the lower bound $kNN^{L}(P)$ derived from $P.^{L}CL$ is larger than or equal to $MaxST(C, q)$, and we can report all objects in C to be results if $kNN^{U}(P)$ derived from $P.^{U}CL$ is smaller than $MinST(C, q)$. Since the functions $MinST$, $TightMinST$ in $P.^{L}CL$ between parent entries are more conservative than those between child entries, i.e., $MinST(C, E_i) {\geq} MinST(P, E_i)$ and $TightMinST(C, E_i) \geq TightMinST(P, E_i)$, thus the lower bound $kNN^{L}(C)$ derived from $C.^{L}CL$ is larger than or equal to $kNN^{L}(P)$ derived from $P.^{L}CL$. Thus if $MaxST(C, q) \leq kNN^{L}(P)$, then $MaxST(C, q) \leq kNN^{L}(C)$. Therefore we can safely prune child entry C. It is similar for the inheritance of upper bound contribution list. Since the function $MaxST$ between parent entries is more conservative than that between child entries, thus $kNN^{U}(C) \leq kNN^{U}(P)$. If $MinST(C, q)$ $> kNN^{U}(P)$ inherited from parent P, then $MinST(C, q) > kNN^{U}(C)$, and thus we can add child entry C as a result safely. Therefore, Claim 3 holds.

CLAIM 4. *"Lazy travel down" technology used in Algorithm 3 is correct.*

PROOF: We need to prove that entries can be safely pruned or be reported to be results without accessing the subtrees of the pruned entries due to "lazy travel down". That is, in Line 8 of Algorithm 3, we do not need to access the pruned entries to affect entries being processed currently. Obviously it does not influence an entry that is not part of results, since we can prune an entry E as long as we find at least k objects that are more similar than the query object. Meanwhile, entries E can be safely reported as results if it satisfies the condition of $MinST(E, q){>}kNN^{U}(E)$ even without accessing the subtrees of the pruned entries by "lazy traveled down", as according to the algorithm, all the pruned entries must be already used to compute the upper bound $kNN^{U}(E)$ of the entry E or the upper bound $kNN^{U}(A)$ of E's ancestor entry A. In particular, if the pruned entries are already used to compute the bounds of entry E, then it holds trivially. If the pruned entries are used to compute the bounds of the entry A, it is also correct due to the "inherit" technology. That is, the inherited value $kNN^{U}(A)$ is larger than or equal to $kNN^{U}(E)$. Thus if $MinST(E, q) > kNN^{U}(A)$, then $MinST(E, q) > kNN^{U}(E)$, and then entry E can be safely reported as results. Therefore, Claim 4 holds.

Completeness: All objects which can not be safely pruned or reported as results, are appended to the candidate object list COL (Line 19 in Algorithm 3). In the $FinalVerification$ procedure, all the candidate objects can be determined whether they are results through traveling down the pruned entries. It is because that even in the worst case, we can access all the objects in the subtree of the pruned entries to determine each candidate object if it is an answer in $IsHitOrDrop$ (Line 6 in Algorithm 4) while $MinST$ and $MaxST$ between two database objects are equal. Thus our algorithm is complete, *i.e.*, it can return all the RSKkNN data points.

Hence, Theorem 1 is true. □

6.3. Performance Analysis

In this subsection, we propose an analytical model to estimate the cost of the RSKkNN queries and theoretically analyze the performance of the RSKkNN algorithm based on the IUR-tree. The number of accessed nodes in the IUR-tree is computed in Theorem 2.

Theorem 2. *Assume that the locations of N objects are uniformly distributed in 2-dimension space, and the word frequencies in each object follow the Zipf distribution. With high probability, the number of IUR-tree index nodes accessed using the $RSKkNN$ algorithm is $\mathcal{O}(f \log_f N)$, where f is the fanout of the IUR-tree.*

To show the superiority of the RSKkNN search algorithm, recall the baseline method that is discussed in Section 4, which computes the top-k spatial-textual nearest neighbor objects using the threshold algorithm [Fagin et al. 2003]. The computational cost of the threshold algorithm is $\sqrt{kN}$ [Fagin et al. 2003]. Given that one needs to check whether query object q is one of the top-k nearest neighbors for each object, the overall computational cost of the baseline method is $O(N\sqrt{kN})$, which is much higher than that of the RSKkNN algorithm.

We proceed to introduce the analytical model and prove Theorem 2 in details in this section.

6.3.1. Analysis Model Settings. Following the assumption in [Theodoridis and Sellis 1996], we assume the locations of N objects are uniformly distributed in 2-dimension space. The word frequencies of objects follow the Zipf distribution. Specifically, we assume that there is a word pool with M distinct words whose frequencies are assumed to follow the Zipf distribution, i.e., the frequency of the k-th most popular word is $\frac{1/k^s}{\sum_{i=1}^{M}(1/i^s)}$, where $k \in [1, M]$, and s is a parameter characterizing the distribution. Then we randomly select m words from the word pool for each object. Our goal is to estimate the expected number DA of the IUR-tree by Algorithms 3 and 4.

To facilitate the analysis, suppose that N objects are indexed in an IUR-tree with the height h (the root is assumed to be at level h and leaf-nodes are assumed to be at level 1), and let NN_l denote the number of index nodes at level l, and let P_l and A_l be the number of index nodes that can be pruned and can be reported as results at level l, respectively. Let f denote the fanout of the IUR-tree. Thus, R_l, which is the number of entries that should be processed at level l is at least:

$$R_l = NN_l - f(P_{l+1} + A_{l+1}) \tag{20}$$

Let X_l denote the number of additional entries that are accessed to compute the lower and upper kNN bounds of entries at level l. Therefore, at level l in the IUR-tree, the number of accessed index nodes is:

$$DA_l = R_l + X_l \tag{21}$$

Thus the total number of node accesses in the IUR-tree can be derived as:

$$DA = 1 + \sum_{l=1}^{h-1}(NN_l - f(P_{l+1} + A_{l+1}) + X_l) \tag{22}$$

The height h of an R-tree with the fanout f that stores N data entries is given by Eqn(23) in [Faloutsos et al. 1987]. The number of index nodes at level l is given in Eqn(24). Note that the number of objects in each node at level l is f^l.

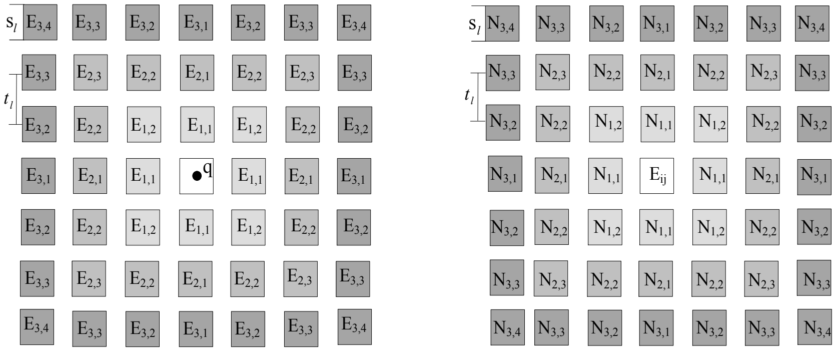
$$h = 1 + \lceil \log_f \frac{N}{f} \rceil \tag{23}$$

$$NN_l = \frac{N}{f^l} \tag{24}$$

In the following, we first estimate P_l in Section 6.3.2 (resp. A_l in Section 6.3.3), the number of index nodes pruned (resp. reported as results) at level l in the IUR-tree by considering only the spatial information, denoted by PS_l and AS_l, respectively, and then show how to take into account the textual information in Section 6.3.4.

6.3.2. Estimation of the number of pruned entries at level l without texts. Figure 9 illustrates the layout of entries and the query q at level l. Let s_l denote the side extent of the MBR of an entry at level l and let t_l be the distance between the centers of two consecutive MBRs at level l. As mentioned above, f denotes the fanout of the IUR-tree. Subsequently, Equation (25), which describes the relationships among s_{l+1}, t_l and f, are given in [Theodoridis and Sellis 1996]. We then derive the value of s_l in Eqn(26).

$$s_{l+1} = (f^{\frac{1}{2}} - 1)t_l + s_l \ \ and \ \ t_l = (\frac{f^l}{N})^{\frac{1}{2}} \tag{25}$$


s_l
t_l
E3,4 E3,3 E3,2 E3,1 E3,2 E3,3 E3,4
E3,3 E2,3 E2,2 E2,1 E2,2 E2,3 E3,3
E3,2 E2,2 E1,2 E1,1 E1,2 E2,2 E3,2
E3,1 E2,1 E1,1 q E1,1 E2,1 E3,1
E3,2 E2,2 E1,2 E1,1 E1,2 E2,2 E3,2
E3,3 E2,3 E2,2 E2,1 E2,2 E2,3 E3,3
E3,4 E3,3 E3,2 E3,1 E3,2 E3,3 E3,4
s_l
t_l
N3,4 N3,3 N3,2 N3,1 N3,2 N3,3 N3,4
N3,3 N2,3 N2,2 N2,1 N2,2 N2,3 N3,3
N3,2 N2,2 N1,2 N1,1 N1,2 N2,2 N3,2
N3,1 N2,1 N1,1 Eij N1,1 N2,1 N3,1
N3,2 N2,2 N1,2 N1,1 N1,2 N2,2 N3,2
N3,3 N2,3 N2,2 N2,1 N2,2 N2,3 N3,3
N3,4 N3,3 N3,2 N3,1 N3,2 N3,3 N3,4

(a) Layout of entries $E_{i,j}$ and query object q, where $E_{i,j}$ denotes the entries that are at the i-th layer around q and the j-th minimal distance to entry q. Note that $E_{i,j}$ may refer to multiple entries. For example there are 4 entries denoted by $E_{2,1}$, and 8 entries denoted by $E_{2,2}$.

(b) Layout of entries $E_{i,j}$ (in Figure 9(a)) and $N_{i,j}$, where $N_{i,j}$ is an entry that is at the i-th layer around $E_{i,j}$ and the j-th minimal distance to entry $E_{i,j}$. Note that $N_{i,j}$ refers multiple entries, all of which has the same distance from $E_{i,j}$.

Fig. 9. Layout of query q and entries at level l

$$s_l \;=\; t_l - (\frac{1}{N})^{\frac{1}{2}} \tag{26}$$

As shown in Figure 9(a), intuitively, entries that are far away from the query object q are more likely to be pruned. The number of pruned entries, PS_l, is formally computed in Lemma 6 as follows.

LEMMA 6. *The number PS_l of pruned entries at level l in the IUR-tree (by taking into account only spatial information) is : $PS_l = \frac{N}{f^l} - \left\lceil (8\sqrt{2} - 4)r^2 + 20\sqrt{2} * r + 8\sqrt{2} + 1 \right\rceil$, where $r = \left\lceil 0.25\sqrt{\frac{4k+4}{f^l} + 13} - 0.5 \right\rceil$.*

PROOF. Recall that in our algorithm, we can safely prune an entry $E_{i,j}$ in Figure 9(a) if $\forall o \in E_{i,j}$, there are at least k objects o' such that $MinS(E_{i,j}, q) > dist(o, o')$, where $MinS(E_{i,j}, q)$ is the minimal spatial distance between entry $E_{i,j}$ and q. Thus, we need first to estimate a distance, denoted by $MaxkNN$, such that there are at least k objects whose distance to $E_{i,j}$ is no greater than $MaxkNN$.

As shown in Figure 9(b), which gives the layout of the entry $E_{i,j}$ and the surrounded entries $N_{i,j}$, we need to identify the layer r such that there are at least $k - (f^l - 1)$ objects in entries $N_{i,j}$, $i{\le}r$, $j{\le}r$, around $E_{i,j}$. Subtracting $f^l - 1$ is because that for any object o in $E_{i,j}$, there are already $|E_{i,j}| - 1$ other objects within $E_{i,j}$. We show a table in Figure 10(a), where each cell at row i and column j contains a binary-tuple $<A_{i,j}, B_{i,j}>$, where $A_{i,j}$ is the maximal spatial distance $MaxS(E_{i,j}, N_{i,j})$ between entry $E_{i,j}$ and $N_{i,j}$; and $B_{i,j}$ is the number of objects in $N_{i,j}$. For example, for the cell $< A_{1,1}, B_{1,1} >$, $A_{1,1} = \sqrt{s_l^2 + (s_l + t_l)^2}$ is the maximal distance between entries $N_{1,1}$ and $E_{i,j}$, and $B_{1,1} = 4f^l$ is the total number of objects in the 4 entries $N_{1,1}$. Note that the number of objects in each entry at level l is f^l. Given any three distance values $A_{i,j}$, $A_{i,j+1}$ and $A_{i+1,j}$ in three adjacent cells of Figure 10(a), it is easy to prove that $A_{i,j} < A_{i,j+1}$ and $A_{i,j} < A_{i+1,j}$ hold.

Thus, as shown in Eqn(27), we can get the value of r in $A_{r,r+1}$ (i.e., $N_{r,r+1}$) such that there are at least $k - (f^l - 1)$ objects in entries $N_{i,j}$, $i{\le}r$, $j{\le}r$, around $E_{i,j}$:

	1	2	3	4
1	$<\sqrt{{s_l}^2+(s_l+t_l)^2},4f^l>$	$<\sqrt{(s_l+t_l)^2+(s_l+t_l)^2},4f^l>$		
2	$<\sqrt{{s_l}^2+(s_l+2t_l)^2},4f^l>$	$<\sqrt{(s_l+t_l)^2+(s_l+2t_l)^2},8f^l>$	$<\sqrt{(s_l+2t_l)^2+(s_l+2t_l)^2},4f^l>$	
3	$<\sqrt{{s_l}^2+(s_l+3t_l)^2},4f^l>$	$<\sqrt{(s_l+t_l)^2+(s_l+3t_l)^2},8f^l>$	$<\sqrt{(s_l+2t_l)^2+(s_l+3t_l)^2},8f^l>$	$<\sqrt{(s_l+3t_l)^2+(s_l+3t_l)^2},4f^l>$

(a) Summary for the maximal distances between the entry $E_{i,j}$ and entries $N_{i,j}$ around $E_{i,j}$ in Figure 9(b) and the numbers of objects in $N_{i,j}$. Assume that the entry $N_{i,j}$ at row i and column j in the table is $<A_{i,j}, B_{i,j}>$, where $A_{i,j}$ is the maximal spatial distance $MaxS(E_{i,j}, N_{i,j})$; and $B_{i,j}$ is the number of objects in entry $N_{i,j}$.

	1	2	3	4
1	$<\sqrt{(t_l-\frac{1}{2}s_l)^2},4>$	$<\sqrt{(t_l-\frac{1}{2}s_l)^2+(t_l-\frac{1}{2}s_l)^2},4>$		
2	$<\sqrt{(2t_l-\frac{1}{2}s_l)^2},4>$	$<\sqrt{(t_l-\frac{1}{2}s_l)^2+(2t_l-\frac{1}{2}s_l)^2},8>$	$<\sqrt{(2t_l-\frac{1}{2}s_l)^2+(2t_l-\frac{1}{2}s_l)^2},4>$	
3	$<\sqrt{(3t_l-\frac{1}{2}s_l)^2},4>$	$<\sqrt{(t_l-\frac{1}{2}s_l)^2+(3t_l-\frac{1}{2}s_l)^2},8>$	$<\sqrt{(2t_l-\frac{1}{2}s_l)^2+(3t_l-\frac{1}{2}s_l)^2},8>$	$<\sqrt{(3t_l-\frac{1}{2}s_l)^2+(3t_l-\frac{1}{2}s_l)^2},4>$

(b) Summary for the minimal distance between entries $E_{i,j}$ and query q in Figure.9(a) and the number of entries $E_{i,j}$. Let $<A'_{i,j}, B'_{i,j}>$ denote the entry $E_{i,j}$ at row i and column j, where $A'_{i,j}$ is the minimal spatial distance $MinS(E_{i,j}, q)$ between entry $E_{i,j}$ and query q; $B'_{i,j}$ is the number of entries $E_{i,j}$ s.t. $MinS(E_{i,j}, q) = A'_{i,j}$.

Fig. 10. Illustration for the maximal spatial distances between entries and minimal spatial distances between query object q and entries at level l.

$$\sum_{i=1}^{r-1}(2*4f^l+8f^l(i-1))+4f^l+8f^l(r-1)=k-(f^l-1)$$

$$\Longrightarrow r=\left\lceil 0.25\sqrt{\frac{4k+4}{f^l}+13}-0.5\right\rceil \tag{27}$$

We then identify entries $E_{i,j}$ in Figure 9(a) which can safely be pruned, i.e., finding entries whose minimal spatial distances to query object q are larger than the value of $A_{r,r+1}$. Intuitively, entries that are farther away from q are more likely to be pruned. Figure 10(b) illustrates the minimal distances $MinS(E_{i,j}, q)$ between entry $E_{i,j}$ and query object q. In particular, assume that $<A'_{i,j}, B'_{i,j}>$ is an entry at row i and column j in Figure 10(b), then $A'_{i,j}$ is the minimal spatial distance between entry $E_{i,j}$ and query q; $B'_{i,j}$ is the number of entries $E_{i,j}$ s.t. $MinS(E_{i,j}, q) = A'_{i,j}$. For example, for the cell $< A'_{2,2}, B'_{2,2} >$, $A'_{2,2} = \sqrt{(t_l - \frac{1}{2}s_l)^2 + (2t_l - \frac{1}{2}s_l)^2}$ is the minimal distance between entries $E_{2,2}$ and q, and $B'_{2,2} = 8$ is the number entries denoted by $E_{2,2}$, i.e., entries whose minimal distance to q is $A'_{2,2}$. Given any three distance values $A'_{i,j}$, $A'_{i,j+1}$ and $A'_{i+1,j}$ in three adjacent cells in Figure 10(b), it is easy to verify that $A'_{i,j+1} > A'_{i,j}$, $A'_{i+1,j} > A'_{i,j}$ and $A'_{\lceil\sqrt{2}*i\rceil,1} > A'_{i,i+1}$ hold. Further, we can explore the relationship of distance values in Figures 10(a) and (b) as follows. Let $A_{i,j}$ denote the distance value at row i column j in Figure 10(a), and let $A'_{i,j}$ denote the distance value in Figure 10(b). We can see $A'_{i+2,j+2} > A_{i,j}$. Therefore, given the distance $A_{r,r+1}$ in Figure 10(a), then the values $A_{i,j}$ in Figure 10(b), where ($i{\geq}r$+2 and $j{\geq}r$+2) or ($i{\geq}\lceil\sqrt{2}(r+2)\rceil$ and $j{<}r+2$), are larger than $A'_{r,r+1}$.

Therefore, to compute the number of pruned index nodes PS_l, we use the total number of entries at level l, $\frac{N}{f^l}$, to subtract the number of entries that cannot be pruned, i.e., the entry containing q in Figure 10(b), and entries $E_{i,j}$ where ($i{<}r$+2 and $j{<}r$+2) or ($i{<}\lceil\sqrt{2}(r+2)\rceil$ and $j{<}r+2$), that is:

$$\begin{aligned} PS_l &= \frac{N}{f^l} - \left[1 + \sum_{i=1}^{r}\big(8 + 8(i-1)\big) + 4 + 8r + \big(\sqrt{2}(r+2) - (r+1)\big)\big(4+8r\big)\right] \\ &= \frac{N}{f^l} - \left[(8\sqrt{2}-4)r^2 + 20\sqrt{2} * r + 8\sqrt{2} + 1\right] \end{aligned} \tag{28}$$

Together with Eqn(27) and Eqn(28), Lemma 6 holds. □

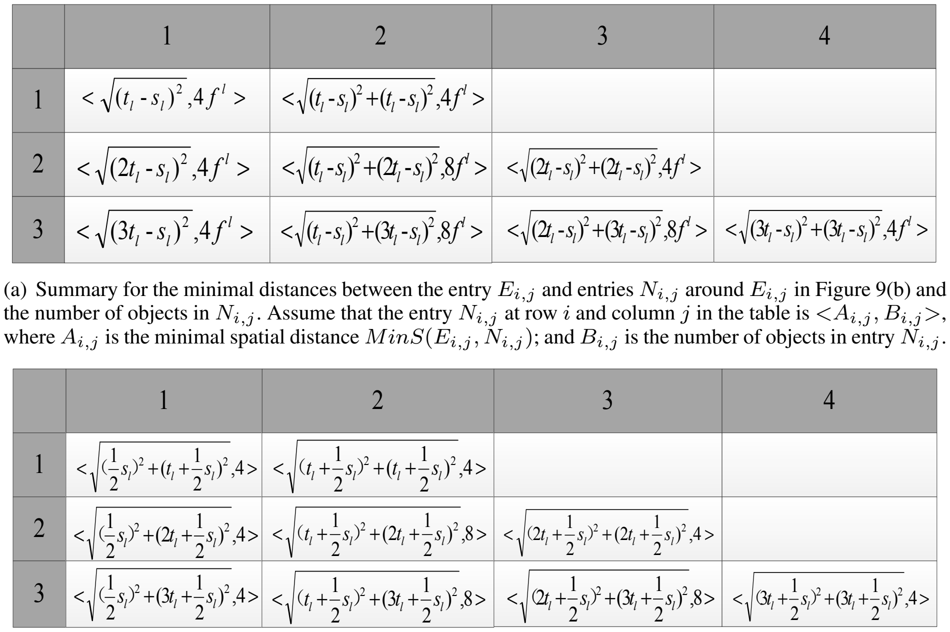
6.3.3. Estimation for the number of entries reported as results at Level l without texts. As shown in Figure 9(a), intuitively, entries that are closer to the query object q are more likely to be reported as results. The number AS_l of entries that can be reported as results is given in Lemma 7.

LEMMA 7. *The number AS_l of entries that can be reported as results at level l in the IUR-tree (by taking into account only spatial information) is $AS_l = 4p^2 - 12p + 5$, where $p = \lceil 0.387 + \sqrt{\frac{0.137(k+1)}{f^l} + 1.617}\rceil$.*

PROOF. Recall that in our algorithm, we can safely report an entry $E_{i,j}$ in Figure 9(a) to be a result entry (i.e., all the objects in $E_{i,j}$ are results) if $\forall o \in E_{i,j}$, there are at most k objects o' such that $MaxS(E_{i,j}, q) < dist(o, o')$, where $MaxS(E_{i,j}, q)$ is the maximal spatial distance between entry $E_{i,j}$ and q. Thus to decide entries $E_{i,j}$ in Figure 9(a) whether to be a result entry, we first need to estimate a distance value, denoted by $MinkNN$, such that within the distance $MinkNN$ to $E_{i,j}$, there are at most k objects, and then determine $E_{i,j}$ whether to be a result entry according to the relationship between the values of $MinkNN$ and $MaxS(E_{i,j}, q)$.

We first compute the value of $MinkNN$. As shown in Figure 9(b), we need to identify an entry $N_{p,p+1}$ such that there are at most $k - (f^l - 1)$ objects in entries $N_{i,j}$, $i{\leq}p$, $j{\leq}p$, around $E_{i,j}$. Again subtracting $f^l - 1$ is because that for any object o in $E_{i,j}$, there are already $|E_{i,j}| - 1$ other objects within $E_{i,j}$. Thus to facilitate identify $N_{p,p+1}$, we need to compute the minimal spatial distances between the entry $E_{i,j}$ in Figure 9(b) and entries $N_{i,j}$ around $E_{i,j}$. As illustrated in Figure 11(a), given any three distance values $A_{i,j}$, $A_{i,j+1}$ and $A_{i+1,j}$ in three adjacent cells, we have $A_{i,j} < A_{i,j+1}$ and $A_{i,j} < A_{i+1,j}$. Further, given any a row number i, we have $A_{\lceil\sqrt{2}*i\rceil,1} > A_{i,i+1}$ as well. Therefore, in Figure 11(a), the values $A_{i,j}$, ($i \leq p$ and $j \leq p$) or ($i < \sqrt{2} * p$ and $j \leq p$), are smaller than $A_{p,p+1}$. Thus as shown in Eqn(29), we can compute the value of p in $A_{p,p+1}$ ($MinkNN$ = $A_{p,p+1}$) such that there are at most $k - (f^l - 1)$ objects in entries $N_{i,j}$, ($i \leq p$ and $j \leq p$) or ($i < \sqrt{2} * p$ and $j \leq p$), around $E_{i,j}$.

$$\begin{aligned} &\sum_{i=1}^{p-1}\big(8f^l + 8f^l(i-1)\big) + 4f^l + 8f^l(p-1) + (\sqrt{2}p - p)\big(4f^l + 8f^l(p-1)\big) = k - (f^l - 1) \\ &\Longrightarrow p = \lceil 0.387 + \sqrt{\frac{0.137(k+1)}{f^l} + 1.617}\rceil \end{aligned} \tag{29}$$



	1	2	3	4
1	$<\sqrt{(t_l - s_l)^2}, 4f^l>$	$<\sqrt{(t_l - s_l)^2 + (t_l - s_l)^2}, 4f^l>$		
2	$<\sqrt{(2t_l - s_l)^2}, 4f^l>$	$<\sqrt{(t_l - s_l)^2 + (2t_l - s_l)^2}, 8f^l>$	$<\sqrt{(2t_l - s_l)^2 + (2t_l - s_l)^2}, 4f^l>$	
3	$<\sqrt{(3t_l - s_l)^2}, 4f^l>$	$<\sqrt{(t_l - s_l)^2 + (3t_l - s_l)^2}, 8f^l>$	$<\sqrt{(2t_l - s_l)^2 + (3t_l - s_l)^2}, 8f^l>$	$<\sqrt{(3t_l - s_l)^2 + (3t_l - s_l)^2}, 4f^l>$

(a) Summary for the minimal distances between the entry $E_{i,j}$ and entries $N_{i,j}$ around $E_{i,j}$ in Figure 9(b) and the number of objects in $N_{i,j}$. Assume that the entry $N_{i,j}$ at row i and column j in the table is $<A_{i,j}, B_{i,j}>$, where $A_{i,j}$ is the minimal spatial distance $MinS(E_{i,j}, N_{i,j})$; and $B_{i,j}$ is the number of objects in entry $N_{i,j}$.

	1	2	3	4
1	$<\sqrt{(\frac{1}{2}s_l)^2 + (t_l + \frac{1}{2}s_l)^2}, 4>$	$<\sqrt{(t_l + \frac{1}{2}s_l)^2 + (t_l + \frac{1}{2}s_l)^2}, 4>$		
2	$<\sqrt{(\frac{1}{2}s_l)^2 + (2t_l + \frac{1}{2}s_l)^2}, 4>$	$<\sqrt{(t_l + \frac{1}{2}s_l)^2 + (2t_l + \frac{1}{2}s_l)^2}, 8>$	$<\sqrt{(2t_l + \frac{1}{2}s_l)^2 + (2t_l + \frac{1}{2}s_l)^2}, 4>$	
3	$<\sqrt{(\frac{1}{2}s_l)^2 + (3t_l + \frac{1}{2}s_l)^2}, 4>$	$<\sqrt{(t_l + \frac{1}{2}s_l)^2 + (3t_l + \frac{1}{2}s_l)^2}, 8>$	$<\sqrt{(2t_l + \frac{1}{2}s_l)^2 + (3t_l + \frac{1}{2}s_l)^2}, 8>$	$<\sqrt{(3t_l + \frac{1}{2}s_l)^2 + (3t_l + \frac{1}{2}s_l)^2}, 4>$

(b) Summary for the maximal distances between entries $E_{i,j}$ and query object q and the number of entries $E_{i,j}$ in Figure 9(a). Let $<A'_{i,j}, B'_{i,j}>$ denote the entry $E_{i,j}$ at row i and column j, where $A'_{i,j}$ is the maximal spatial distance $MaxS(E_{i,j}, q)$ between entry $E_{i,j}$ and query q; $B'_{i,j}$ is the number of entries $E_{i,j}$ s.t. $MaxS(E_{i,j}, q) = A'_{i,j}$.

Fig. 11. Illustration for the minimal spatial distances between entries and maximal spatial distances between query object q and entries at level l.

We then identify which entries $E_{i,j}$ in Figure 9(a) to be result entries, i.e., find entries whose maximal spatial distances to query object q are definitely smaller than value of $MinkNN$. Intuitively, entries that are closer to q are more likely to be result entries. Figure 11(b) illustrates the maximal distances $MaxS(E_{i,j}, q)$ between entry $E_{i,j}$ and query object q. In particular, assume that $<A'_{i,j}, B'_{i,j}>$ is an entry at row i and column j in Figure 11(b), then $A'_{i,j}$ is the maximal spatial distance between $E_{i,j}$ and q; $B'_{i,j}$ is the number of entries $E_{i,j}$ s.t. $MaxS(E_{i,j}, q) = A'_{i,j}$. Given any three distance values $A'_{i,j}$, $A'_{i,j+1}$ and $A'_{i+1,j}$ in three adjacent cells in Figure 10(b), we have $A'_{i,j+1} > A'_{i,j}$ and $A'_{i+1,j} > A'_{i,j}$. Furthermore, we can explore the relationship of distance values in Figures 11(a) and (b) as follows: Let $A_{i,j}$ denote the distance value at row i column j in Figure 11(a), and let $A'_{i,j}$ denote the distance value in Figure 11(b), we have $A'_{i-2,j-2} < A_{i,j}$.

Based the observations above, we can derive the following: Given the distance $A_{p,p+1}$ (whose value is $MinkNN$) in Figure 11(a), then the values $A_{i,j}$ in Figure 11(b) satisfying ($i \leq p$-2 and $j \leq p$-2) are smaller than $A'_{p,p+1}$. Therefore, any entry $E_{i,j}$ in Figure 11(b) can be reported as a result entry if ($i \leq p$-2 and $j \leq p$-2). Thus we can derive the number of index nodes AS_l that can be reported as results:

$$AS_l = 1 + \sum_{i=1}^{p-3} (2 * 4 + 8(i-1)) + 4 + 8(p-2-1) = 4p^2 - 12p + 5 \tag{30}$$

Together with Eqn(29) and Eqn(30), Lemma 7 holds. □

6.3.4. Estimation for the number of accessed nodes with texts. We now describe how to take into account the textual information. The main idea is to investigate how the textual information can influence the number of pruned and reported index nodes.

LEMMA 8. *Considering both spatial and textual information, the number P_l of pruned entries at level l in the IUR-tree satisfies: $P_l \geq PS_l - 8^{\frac{\frac{1-\alpha}{\alpha} * \frac{\psi_s - \varphi_s}{\psi_t - \varphi_t} * [MaxT(E,q) - MinT(E,E')]}{t_l}}$, where $MaxT(E, q)$ is the maximum textual relevance between an entry E at level l and query object q, $MinT(E, E')$ is the minimum textual relevance between two entries E and E' at level l in the IUR-tree, and t_l is given in Eqn(25).*

PROOF. Recall Line 10 in Algorithm 4. We can prune an entry E if and only if:

$$\begin{aligned}
& kNN^L(E) \geq MaxST(E, q) \\
\Rightarrow\ & \alpha(1 - \frac{MaxS(E, E') - \varphi_s}{\psi_s - \varphi_s}) + (1 - \alpha)\frac{MinT(E, E') - \varphi_t}{\psi_t - \varphi_t} \\
& \geq \alpha(1 - \frac{MinS(E, q) - \varphi_s}{\psi_s - \varphi_s}) + (1 - \alpha)\frac{MaxT(E, q) - \varphi_t}{\psi_t - \varphi_t} \\
\Rightarrow\ & MinS(E, q) \geq MaxS(E, E') + \frac{1 - \alpha}{\alpha} * \frac{\psi_s - \varphi_s}{\psi_t - \varphi_t} * [MaxT(E, q) - MinT(E, E')]
\end{aligned}$$

The above formula shows that the number of pruned nodes should be reduced due to the textual values. In particular, consider Figure 10(b) again, since we can prove that the gap between two adjacent cells with respect to the values $A_{i,j}$ is no less than t_l, and there are at most 8 entries in any cell of Figure 10(b) (i.e., $\forall B'_{i,j} \leq 8$), we can derive the number of pruned entries has been changed as shown in Lemma 8. Hence Lemma holds. □

LEMMA 9. *Considering both spatial and textual information, at level l in the IUR-tree, the number A_l of entries that are reported as results satisfies that: $A_l \geq AS_l + 4^{\frac{\frac{1-\alpha}{\alpha} * \frac{\psi_s - \varphi_s}{\psi_t - \varphi_t} * [MinT(E,q) - MaxT(E,E')]}{t_l}}$, where $MinT(E, q)$ is the minimum textual relevance between an entry E at level l and query object q, $MaxT(E, E')$ is the maximum textual relevance between two entries E and E' at level l in the IUR-tree, and t_l is given in Eqn(25).*

PROOF. Recall Line 14 in Algorithm 4. We report entry E as a result entry if and only if:

$$\begin{aligned}
& kNN^U(E) < MinST(E, q) \\
\Rightarrow\ & \alpha(1 - \frac{MinS(E, E') - \varphi_s}{\psi_s - \varphi_s}) + (1 - \alpha)\frac{MaxT(E, E') - \varphi_t}{\psi_t - \varphi_t} \\
& < \alpha(1 - \frac{MaxS(E, q) - \varphi_s}{\psi_s - \varphi_s}) + (1 - \alpha)\frac{MinT(E, q) - \varphi_t}{\psi_t - \varphi_t} \\
\Rightarrow\ & MaxS(E, q) < MinS(E, E') + \frac{1 - \alpha}{\alpha} * \frac{\psi_s - \varphi_s}{\psi_t - \varphi_t} * [MinT(E, q) - MaxT(E, E')]
\end{aligned}$$

Further, there are at least 4 entries in any cell of Figure 11(b) (i.e., $\forall B'_{i,j} \geq 4$), and thus A_l can be derived as that in Lemma 9. □

LEMMA 10. *Considering spatial and textual information, under a high probability lager than $1 - \frac{1}{\sqrt{2^s - 1}}$, the number R_l of the entries that need to be processed at level l in the IUR-tree is:*

$$\begin{aligned}
R_l \ \leq\ & \frac{1.28(k+1)}{f^l} + 5.243f\sqrt{\frac{4(k+1)}{f^{l+1}} + 13} + 8.904f\sqrt{\frac{0.137(k+1)}{f^{l+1}} + 1.617} - 1.48f \\
& + 8f^{\frac{\frac{1-\alpha}{\alpha} * \frac{\psi_s - \varphi_s}{\psi_t - \varphi_t} * \frac{(2\zeta(s)^{2mf^{i+1}} - 2)(2^s - 1) + \frac{2}{m}}{\zeta(s)^{2mf^{i+1}}\sqrt{2^s - 1}}}{t_{i+1}}}
\end{aligned}$$

where $\zeta(s) = \lim_{M \to \infty} \sum_{i=1}^{M}(1/i^s)$, *which is known as Riemann's zeta function [Titchmarsh 2005],* s *is the parameter in Zipf distribution assumption.*

PROOF. According to Lemmas 6, 7, 8, and 9, and Eqn(20), the number R_l of the entries which needs to be processed at level l is:

$$\begin{aligned} R_l &= NN_l - f(P_{l+1} + A_{l+1}) \\ &= \frac{1.28(k+1)}{f^l} + 5.243f\sqrt{\frac{4(k+1)}{f^{l+1}+13}} + 8.904f\sqrt{\frac{0.137(k+1)}{f^{l+1}} + 1.617} - 1.48f \\ &\quad + 8f\frac{\frac{1-\alpha}{\alpha} * \frac{\psi_s - \varphi_s}{\psi_t - \varphi_t} * \big(MaxT(E,q) - MinT(E,E') + MaxT(E,E') - MinT(E,q)\big)}{t_{l+1}} \\ &\leq \frac{1.28(k+1)}{f^l} + 5.243f\sqrt{\frac{4(k+1)}{f^{l+1}+13}} + 8.904f\sqrt{\frac{0.137(k+1)}{f^{l+1}} + 1.617} - 1.48f \\ &\quad + 8f\frac{\frac{1-\alpha}{\alpha} * \frac{\psi_s - \varphi_s}{\psi_t - \varphi_t} * \big(2 - MinT(E,E') - MinT(E,q)\big)}{t_{l+1}} \end{aligned} \tag{31}$$

In the following, we proceed to estimate the values of $MinT(E,E')$ and $MinT(E,q)$ to compute R_l.

Recall the assumption about textual distribution that each object contains m words randomly selected from a word pool with M distinct words following Zipf distribution: the frequency of the k-th most popular word w_k is $P_k = \frac{1/k^s}{\sum_{i=1}^{M}(1/i^s)}$. In our algorithm, we estimate $MinT(E,E')$ (resp. $MinT(E,q)$) by Equation (11). For simplicity, assume that all weights are binary, i.e., 0 or 1. Therefore, the key idea in Equation (11) is to estimate the total number of intersection words for each object between two entries E and E' (resp. objects in entry E and query object q). Let $\mathcal{X}_n$ be the random variable representing the number of intersection words of all the n objects. Then the probability that there are x common words appearing in the n objects is no less than that of one special case, where the word w_1 (which is the most popular word in Zipf distribution) is the only common words for all n objects, and the word w_1 appears x times in all the n objects, and the rest $m-x$ words for $n-1$ objects are also word w_1, but the remaining $m-x$ words for the last object are all w_2. Therefore,

$$Pr(\mathcal{X}_n = x) \geq (P_1^x)^n * P_1^{(n-1)(m-x)} * P_2^{m-x} = \frac{2^{sx}}{(\sum_{i=1}^{M}(1/i^s))^{nm} * 2^{sm}}$$

Then the expectation $\mathbb{E}(\mathcal{X}_n)$ of random variable $\mathcal{X}_n$ is:

$$\mathbb{E}(\mathcal{X}_n) = \sum_{x=1}^{m} x * Pr(\mathcal{X}_n = x) \geq \frac{m2^s - m - 1}{(\sum_{i=1}^{M}(1/i^s))^{nm}(2^s-1)} \geq \frac{m2^s - m - 1}{\zeta(s)^{nm} * (2^s - 1)} \tag{32}$$

When $n = 2f^l$, the expectation of the number of intersection words for $2f^l$ objects in entries E and E' is $\frac{2f^l}{\zeta(s)^{2f^l m}} * \frac{m2^s-m-1}{2^s-1}$, and we can get the expectation $\mathbb{E}(MinT(E,E'))$:

$$\mathbb{E}(MinT(E,E')) \leq \frac{m2^s - m - 1}{\zeta(s)^{nm} * (2^s-1)} * \frac{1}{m} = \frac{2^s - 1 - \frac{1}{m}}{\zeta(s)^{2f^l m}(2^s - 1)}. \tag{33}$$

Similarly, we can derive the expectation of $MinT(E,q)$:

$$\mathbb{E}(MinT(E,q)) \leq \frac{2^s - 1 - \frac{1}{m}}{\zeta(s)^{(f^l+1)m}(2^s-1)}. \tag{34}$$

According to the Markov's inequality [DeGroot and Schervish 2004], we have:

$$\begin{aligned} &Pr\left((2 - MinT(E,E') - MinT(E,q)) \geq \frac{(2\zeta(s)^{2f^l m} - 2)(2^s-1) + \frac{2}{m}}{\zeta(s)^{2f^l m}\sqrt{2^s-1}}\right) \\ \leq\ &\frac{2 - \mathbb{E}(MinT(E,E') - \mathbb{E}(MinT(E,q))}{\frac{(2\zeta(s)^{2f^l m} - 2)(2^s-1) + \frac{2}{m}}{\zeta(s)^{2f^l m}\sqrt{2^s-1}}} = \frac{1}{\sqrt{2^s-1}} \end{aligned} \tag{35}$$

Hence, together with Eqn(31) and Eqn(35) the number R_l of the entries which need to be processed at level l can be derived as in Lemma 10, as desired. □

In the following, we estimate the additional number of disk accesses X_l. In order to compute the lower and upper kNN bounds of the rest R_l entries at level l, we may need to visit additional entries at level l that are already pruned or reported as results at the upper levels, besides the R_l entries that are visited. According to the RSKkNN search algorithm, given k, to compute the lower (or upper) bound of entry E at layer l, we visit its top-k most similar objects, which are within the $\lceil \frac{k-(f^l-1)}{f^l} + 1 \rceil$ entries around entry E as shown in Figure 10. Thus the additional number of disk accesses X_l = $\lceil (\frac{k-(f^l-1)}{f^l} + 1) \rceil R_l \leq (\frac{k}{f^l} + 1)R_l$.

Thus, with a probability larger than 1-$\frac{1}{\sqrt{2^s-1}}$, the total expected number of disk accesses DA_l at level l is $X_l + R_l = (\frac{k}{f^l} + 2)R_l$.

Therefore, the number (DA) of index nodes accessed in the IUR-tree using the $RSKkNN$ algorithm is:

$$\begin{aligned} DA &= 1 + \sum_{l=1}^{h-1}(\frac{k}{f^l} + 2)R_l \\ &\leq 1 + 1.28k(k+1)\frac{1-\frac{f^2}{N^2}}{f^2-1} + 13.782k\sqrt{f(k+1)}\frac{1-\frac{f\sqrt{f}}{N\sqrt{N}}}{f\sqrt{f}-1} + 28.746fk\frac{1-\frac{f}{N}}{f-1} \\ &\quad +2.56(k+1)\frac{1-\frac{f}{N}}{f-1} + 27.564\sqrt{f(k+1)}\frac{1-\sqrt{\frac{f}{N}}}{\sqrt{f}-1} + 57.492f\log_f N \\ &\quad +8f\frac{1-\alpha}{\alpha} * \frac{\psi_s - \varphi_s}{\psi_t - \varphi_t} * \frac{(2\zeta(s)^{2mf^2} - 2)(2^s-1) + \frac{2}{m}}{\zeta(s)^{2mf^2}\sqrt{2^s-1}} * \frac{\sqrt{N}(1-\frac{f}{N})}{\sqrt{f}-1} \end{aligned}$$

In particular, if $s \to \infty$, then probability $1 - \frac{1}{\sqrt{2^s-1}} \to 1$, and $\zeta(s) \to 1$ [Titchmarsh 2005]. In addition, if $f \ll N$, together with $s \to \infty$, then, with a high probability, $DA \leq 1 + 1.28k\frac{k+1}{f^2} + 13.782k\frac{\sqrt{(k+1)}}{f} + 28.746k + 2.56\frac{k+1}{f} + 27.564\sqrt{k+1} + 57.492f\log_f N$ = $\mathcal{O}(\frac{k^2}{f^2} + \frac{k\sqrt{k}}{f} + k + f\log_f N)$. Further, assuming constant and small values for f and k, the number (DA) of index nodes accessed in the IUR-tree using the $RSKkNN$ algorithm is $\mathcal{O}(f\log_f N)$, which finally concludes the proof of Theorem 2.

Extension to Cosine Similarities. Using the cosine distance defined in Eqn (5) as the textual similarity measurement, we only need to replace the estimations of $MinT(E,E')$ and $MinT(E,q)$

with the estimations of $MinT_{cos}(E, E')$ and $MinT_{cos}(E, q)$, respectively, in the Eqn(31) in Section 6.3.4. Under the same assumption of the textual distribution given in Section 6.3.1, the expectation of $MinT_{cos}(E, E')$ (resp. $MinT_{cos}(E, q)$) defined in Eqn(19) is the same as the expectation of $MinT(E, E')$ (resp. $MinT(E, q)$) given in Eqn(33) (resp. Eqn(34)). Hence Theorem 2 still holds using cosine as the textual similarity measurement.

7. Refinements for Hybrid Index

Like the R-tree, the IUR-tree is built based on the heuristics of minimizing the area of MBR of nodes. However, the associated texts of the spatial objects in the same MBR can be very different, because the near spatial objects often belong to different specific categories, such as retail, accommodations, restaurants, and tourist attractions. To compute tighter kNN bounds of the entries, we enhance the IUR-tree with text cluster, yielding an index tree called CIUR-tree given in Section 7.1. Then in Section 7.2 we propose a Combined CIUR-tree (called C^{2}IUR-tree), which aims at combining both location and textual information into account during tree construction, by modifying the similarity functions between enclosing rectangles using textual cluster IDs and locations of objects. We also present two optimization methods to improve the search performance based on CIUR-tree and C^{2}IUR-tree in Section 7.3 and Section 7.4, respectively.

7.1. Cluster IUR-tree: CIUR-tree

We propose to use text clustering to enhance IUR-tree. In the pre-processing stage, we group all the database objects into clusters $C_1, \cdots C_n$ according to their text similarities. We extend each IUR-tree node by the cluster information to generate a hybrid tree called Cluster IUR-tree(CIUR-tree). The CIUR-tree is built based on the spatial proximity as does the IUR-tee. However, each node of the CIUR-tree includes a new entry $ClusterList$ in the form of $(ID{:}N)$, where ID is the cluster id and N is the number of objects of cluster ID in the subtree of the node. The $ClusterList$ on the upper layer $CParent$ is the superimposing of that on lower layer $CChild$. That is, $CParent.N = \sum_{j=1}^{M} CChild_j.N$, where M is the number of children of the node.

Similar to the intersection and union textual vectors in IUR-tree, there are intersection and union cluster vectors at each node in CIUR-tree. For each cluster C_i, $CIntVct_i$ and $CUniVct_i$ include the minimal and maximal weights of each word in C_i, respectively. For example, suppose all the objects in Fig.1 are clustered into three clusters: C_1={p_1, p_4, p_5}, C_2={p_2, p_3, p_6} and C_3={p_7, p_8, p_9}, the intersection and union text vectors of which are shown in Fig 12(a). The CIUR-tree is shown in Fig 12(b).

7.2. Combined CIUR-tree: C^{2}IUR-tree

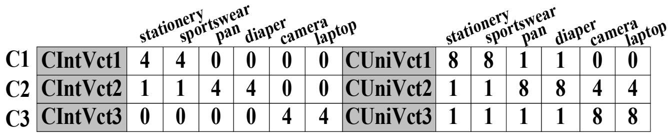
The CIUR tree proposed in the previous section is built on the heuristics of placing nodes that are spatially close in the same MBR. However, the RSKkNN query takes into account both location proximity and text relevancy. In this subsection, we propose the Combined CIUR-tree (called C^{2}IUR-tree), which combines the information about both location and text during tree construction. Specifically, during the construction of the C^{2}IUR-tree, we compute the similarity between two entries by both their spatial proximity and text similarity, which is computed using the cluster IDs in the two entries.

Let E_1,...,E_n be a set of entries. The spatial similarity of a pair of entries, $< E_p, E_q >$, $1 \le p, q \le n$, is defined as follows:

$$\Delta Area(E_p, E_q) = Area(E_{pq}) - Area(E_p) - Area(E_q) \tag{36}$$

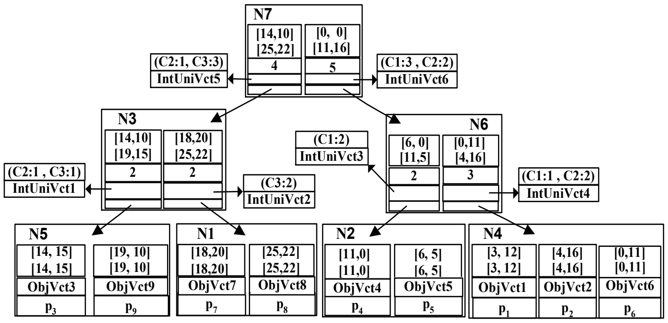
where $Area(E_{pq})$ is the area of the minimum bounding rectangle enclosing E_p and E_q; $Area(E_p)$ and $Area(E_q)$ are the areas of the minimum bounding rectangle enclosing E_p and E_q, respectively. A bigger $\Delta Area(E_p, E_q)$ indicates that the two entries are less similar spatially.

Similarly, the similarity of textual description is defined as follows:



		stationery	sportswear	pan	diaper	camera	laptop		stationery	sportswear	pan	diaper	camera	laptop
C1	CIntVct1	4	4	0	0	0	0	CUniVct1	8	8	1	1	0	0
C2	CIntVct2	1	1	4	4	0	0	CUniVct2	1	1	8	8	4	4
C3	CIntVct3	0	0	0	0	4	4	CUniVct3	1	1	1	1	8	8

(a) Intersection and union text vectors of each cluster



(b) CIUR-tree

Fig. 12. The Cluster IUR-tree of Figure 1

$$\Delta Entropy(E_p, E_q) = Entropy(E_{pq}) - Entropy(E_p) - Entropy(E_q) \tag{37}$$

$$Entropy(E) = -\sum_{i=1}^{n} \frac{cnum_i}{|E|} log \frac{cnum_i}{|E|} \tag{38}$$

where $cnum_i$ is the number of objects of Cluster i in entry E, and $|E|$ is the number of objects in E. $Entropy(E_{pq})$ is the entropy of the cluster IDs vector which combines the two entries E_p and E_q. Larger $\Delta Entropy(E_p, E_q)$ implies that the textual descriptions of two entries are less similar.

Finally, we define the similarity of two entries as follows:

$$Sim(E_p, E_q) = 1 - \left(\beta \left|\frac{\Delta Area(E_p, E_q)}{max\Delta Area}\right| + (1-\beta)\left|\frac{\Delta Entropy(E_p, E_q)}{max\Delta Entropy}\right|\right) \tag{39}$$

where $max\Delta Area$ is the maximum value of $\Delta Area(E_p, E_q)$ for all the pair entries E_p and E_q, $1 \leq p, q \leq n, p \neq q$, which is used for normalization in spatial similarity part; $max\Delta Entropy$ is the maximum value of $\Delta Entropy(E_p, E_q)$ for all the pair entries E_p and E_q, $1 \leq p, q \leq n, p \neq q$, to normalize the textual similarity part. Because $\Delta Area(E_p, E_q)$ and $\Delta Entropy(E_p, E_q)$ can be negative, we use the absolute value to ensure $Sim(E_p, E_q)$ to be positive and monotonic. Parameter β, $0 \leq \beta \leq 1$, is to balance the two similarities. In particular, if $\beta = 1$, then $Sim(E_p, E_q)$ is reduced to the spatial similarity; and if $\beta = 0$, $Sim(E_p, E_q)$ measures only the textual similarity.

While the framework of the algorithm for building C^{2}IUR-tree is similar to that of CIUR-tree, the procedures `ChooseLeaf` and `Split` are different and are presented as follows.

Given a new object, the function `ChooseLeaf` (see Algorithm 5) selects a leaf entry to place it. `ChooseLeaf` travels the tree from the root to a leaf. When it visits an internal node in the tree, it will choose the subtree E with the maximum value of $Sim(E, object)$. Equation 39 can be naturally extended to compute the similarity between an entry and an object. Therefore, the new object will be inserted to the branch which is the most similar to it in terms of the combination of spatial and textual similarity.

ALGORITHM 5: ChooseLeaf ($object$)

```
$N$←root;
while $N$ is not leaf do
    Choose the child entry $E$ of $N$ with max value for $Sim(E, object)$ // defined in Equa. 39;
    $N$←$E$;
Return $N$;
```

ALGORITHM 6: Split (N)

```
$E_{s_1}, E_{s_2} = \underset{E_i, E_j \in N}{argmin}$ Sim($E_i$, $E_j$); // Equation 39
for each entry $E$ in node $N$, where $E \neq E_{s_1}$ and $E \neq E_{s_2}$ do
    if Sim($E$, $E_{s_1}$)$\geq$ Sim($E$, $E_{s_2}$) then
        Classify $E$ as group 1;
    else
        Classify $E$ as group 2;
split $N$ into group 1 and 2
```

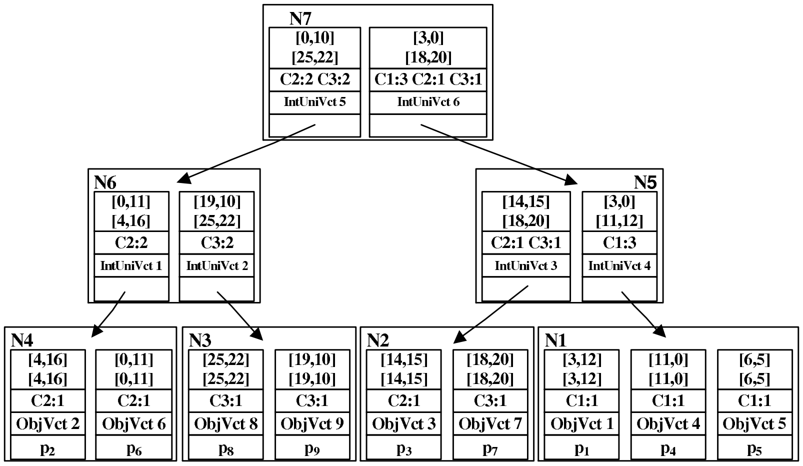


Fig. 13. The example of C^{2}IUR-tree

The function `Split` (see Algorithm 6) is used to split a node N. First we pick two entries E_{s_1} and E_{s_2} in N that have the minimum value of similarity defined in Equation 39, i.e., the two entries have the least similarity from each other. Then for each other entry E in N, if E is closer to E_{s_1} in terms of spatial and textual similarity, E is classified to group 1; otherwise it is in group 2. In this way all entries are split to two groups.

Take the objects in Fig 12(b) for example, the C^{2}IUR-tree is shown in Fig 13. In the CIUR-tree, we group P_1,P_2 and P_6 into a single node, because they are close to each other according to spatial distance. In contrast, in the C^{2}IUR-tree, these objects are partitioned into different nodes; and P_1, P_4 and P_5 are in the same node since they are similar when both textual and spatial information are considered.

7.3. Outlier Detection and Extraction

To develop an optimized algorithm using cluster information in CIUR-tree and C^{2}IUR-tree, one way is to change the order of processing entries in Algorithm 3 to give a priority to entries which have “outliers”. This way, we are more likely to quickly tighten the estimation of low/upper bounds of entries. More precisely, we detect the index node E containing outlier clusters in the following two cases:

Case I: Most objects in $subtree(E)$ can be pruned, but there exist very few objects in $subtree(E)$ that cannot be pruned, and are called "outliers", thus making the whole E non-prunable. More precisely, given a query q, we say one entry E belongs to Case I if 1) $MinST(E,q) < kNN^L(E)$, 2) $MaxST(E,q) > kNN^L(E)$, and 3) there exists a subset of clusters in E, denoted by S_I, such that $\sum_{C_i \in S_I} C_i.N \geq \lambda|E|$, where parameter λ is a threshold close to 1, and $\forall C_i \in S_I$ s.t. $\alpha(1 - \frac{MinS(E,q)-\varphi_s}{\psi_s-\varphi_s}) + (1-\alpha)MaxT(C_i,q) < kNN^L(E)$. The objects that are in E but not in S_I are outliers.

Case II: Most objects in $subtree(E)$ can be reported as answers, but there exist very few objects that are not answers and thus the whole E cannot be reported as a result entry. More precisely, given a query q, an entry E belongs to case II if 1) $MinST(E,q)<kNN^U(E)$, 2) $MaxST(E,q) >kNN^U(E)$, and 3) there exist a subset of clusters in E, denoted by S_{II}, such that $\sum_{C_i \in S_{II}} C_i.N \geq \mu|E|$, where parameter μ is a threshold close to 1, and $\forall C_i \in S_{II}$ s.t. $\alpha(1 - \frac{MaxS(E,q)-\varphi_s}{\psi_s-\varphi_s}) + (1-\alpha)MinT(C_i,q) > kNN^U(E)$.

Having identified entries in Case I or Case II, we process (decompose) them immediately and identify if their subtree entries can be pruned or added as results (without enqueue like normal entries). To implement this optimization for RSKkNN queries, the only change is to replace Line 17 in Algorithm 3 with the following pseudocodes. First, we determine whether the index node E is in Case I or II: if not, then add E into the priority queue U; if yes, then we decompose E by checking whether each subtree entry e of E can be pruned or is a result according to the corresponding relationship between the set C_e of clusters in e and cluster set S_I, S_{II}.

Replace Line 17 in Algorithm 3

```
if ($E$ is in Case $I$ or $II$) then
  for each entry $e \in subtree(E)$
    if $C_e \subset S_I$ then prune $e$; //$C_e$ is the set of clusters in $e$
    else if $C_e \subset S_{II}$ then report $e$ as a result entry;
          else if ($e$ is an index node) then EnQueue($U$,$e$);
                else $COL$.append($e$);
else EnQueue($U$, $E$);
```

7.4. Text-entropy Based Optimization

We proceed to propose the second optimization to improve performance. In particular, we use $TextEntropy$ to depict the distribution of text clusters in an entry of CIUR-tree or C^{2}IUR-tree. Intuitively, the more diverse the clusters are, the larger the $TextEntropy$ of the entry is. The following formula calculates $TextEntropy$ for the leaf and inner nodes recursively.

$$H(E) = \begin{cases} -\sum_{i=1}^{n} \frac{cnum_i}{|E|} log \frac{cnum_i}{|E|} & \text{if } E \text{ is a leaf node;} \\ \sum_{j=1}^{M} \frac{|E.child_j|}{|E|} H(E.child_j) & \text{otherwise.} \end{cases}$$

where $cnum_i$ is the number of objects of Cluster i in entry E, and $|E|$ is the number of objects in E. If E is a leaf node, the $TextEntropy$ describes the distribution of textual cluster in E. If E is an intermediate node, $TextEntropy$ of E is a weighted combination of the $TextEntropy$ of its child entries $E.child_i$.

We use $TextEntropy$ as the priority (key) for the max-priority queue U. If an entry is more diverse in its text description, it has a higher priority to be visited first. By doing so, we expect that decomposing entries/nodes with diverse textual description in sub-entries would reduce the diversity of the entries, and thus would be more likely to enable to quickly tighten the estimation of the lower/upper bounds of similarity between each entry and its kth most similar object through "mutually effect" among entries. In addition, since $TextEntropy$ can be computed offline during the indexing construction, we do not need to access the $ClusterLists$ from disk during the query time. Therefore, $TextEntropy$ based method needs less I/O cost compared to the outlier-detection based optimization

A salient feature of the two above optimizations in Section 7.3 and Section 7.4 is that they are orthogonal and can be combined, as implemented in our experiments.

8. Experimental Studies

We conducted a thorough experimental study to evaluate the efficiency and scalability of our methods in answering RSKkNN queries.

Implemented algorithms We implemented the proposed algorithms based on the IUR-tree as well as the optimizations based on the CIUR-tree: outlier-detection-extraction optimization (ODE-CIUR) and text-entropy optimization (TE-CIUR), and the combination of two optimizations (ODE-TE). We also implemented the search algorithms based on the C^{2}IUR-tree, namely ODE-C^{2}IUR, TE-C^{2}IUR and ODE-TE-C^{2}IUR. In addition, we implemented the two baseline methods discussed in Section 4: the threshold TA-based method and the IR-tree-based method.

Table III. Datasets for the experiments

Statistics	GN	CD	Shop
total # of objects	1,868,821	1,555,209	803,155
total unique words in dataset	222,409	21,578	3933
average # words per object	4	47	45

Datasets and Queries The algorithms are evaluated using three datasets: GeographicNames (GN), CaliforniaDBpedia (CD), and ShopBranches (Shop). These datasets differ from each other in terms of data-size, spatial-distribution, word-cardinality and text-size. Our goal in choosing these diverse sources is to understand the effectiveness and efficiency of our algorithms in different environments. The statistics of each dataset are shown in Table III.

In particular, the GeographicNames dataset (geonames.usgs.gov) is a real-life dataset from the U.S. Board on geographic names with a large number of words to describe the information about each geographic location. The CaliforniaDBpedia dataset combines a real spatial data at California (www.usgs.gov) and a real collection of document abstracts about California in DBpedia (wiki.dbpedia.org/Downloads351). Finally, the ShopBranches is generated from a real-life data describing 955 shop branches and their products. We enlarge the original data by copying and shifting all objects to their neighborhood region while maintaining the distribution of the locations and textual information of the objects. In this way, the size of the data is scaled by up to more than 800 times.

For each dataset, we generated 7 sets of query sets, in each of which the number of keywords is 2, 4, 8, 16, 32, 64 and 128, respectively. Each query set comprises 100 queries, each corresponding to a randomly selected object from the corresponding dataset. We report the average running time of 100 queries for each query set. Note that the tested queries are meaningful in real application scenarios. For example, in ShopBranches data, an RSKkNN query can be used to find shops that will be influenced by a new store outlet. Alternatively, in GeographicNames data, an RSKkNN query can be used to find national parks that will be influenced by a new park with a similar natural landscape.

Setup and metrics We implemented all the algorithms with VC++6.0 on a server with an Intel(R) Core(TM)2 Quad CPU Q8200 @2.33GHz and 4GB of RAM. We implemented the algorithms based on both disk-resident and memory-resident data for the proposed index structures, namely IUR-tree, CIUR-tree and C^{2}IUR-tree. The page size is 4KB and the branch number of each index node is 102.

Both parameters λ and μ in ODE-CIUR are fixed at 0.9 by default. In the CIUR-tree and C^{2}IUR-tree, we cluster the textual vectors of objects into different number of clusters using the *DBSCAN* [Ester et al. 1996] clustering algorithm.

We compare various algorithms with different experimental settings as follows:

parameter k: *1 $\sim$ 128, default 4*
parameter α: *0 $\sim$ 1, default 0.7*
number of query words qw: *1$\sim$128, default 16*
number of clusters: *18$\sim$14337, default 187*

Space requirement The space usage for the structures used by the algorithms is shown in Table IV. The space usage of the CIUR-tree is slightly larger than that of the IUR-tree, since it needs extra space to store the intersection and union vectors for each cluster and the additional new entry "*ClusterList*" on each node. In addition, the space requirement of the C^{2}IUR-tree is also comparable with that of the CIUR-tree. The reason is that the C^{2}IUR-tree requires the same cluster information as does the CIUR-tree. The difference between the CIUR-tree and the C^{2}IUR-tree is that constructing the CIUR-tree is based on the heuristic of minimizing the spatial proximities of objects in CIUR-tree nodes; while the C^{2}IUR-tree is built based on the heuristic of minimizing both spatial and textual similarities of objects enclosed in the C^{2}IUR-tree node. Thus the objects enclosed in a C^{2}IUR-tree node and a CIUR-tree node are different.

Table IV. Sizes of indexing structures

Data	IUR-tree	CIUR-tree	C^{2}IUR-tree
GN	264MB	306MB	302MB
CD	218MB	237MB	231MB
Shop	210MB	288MB	283MB

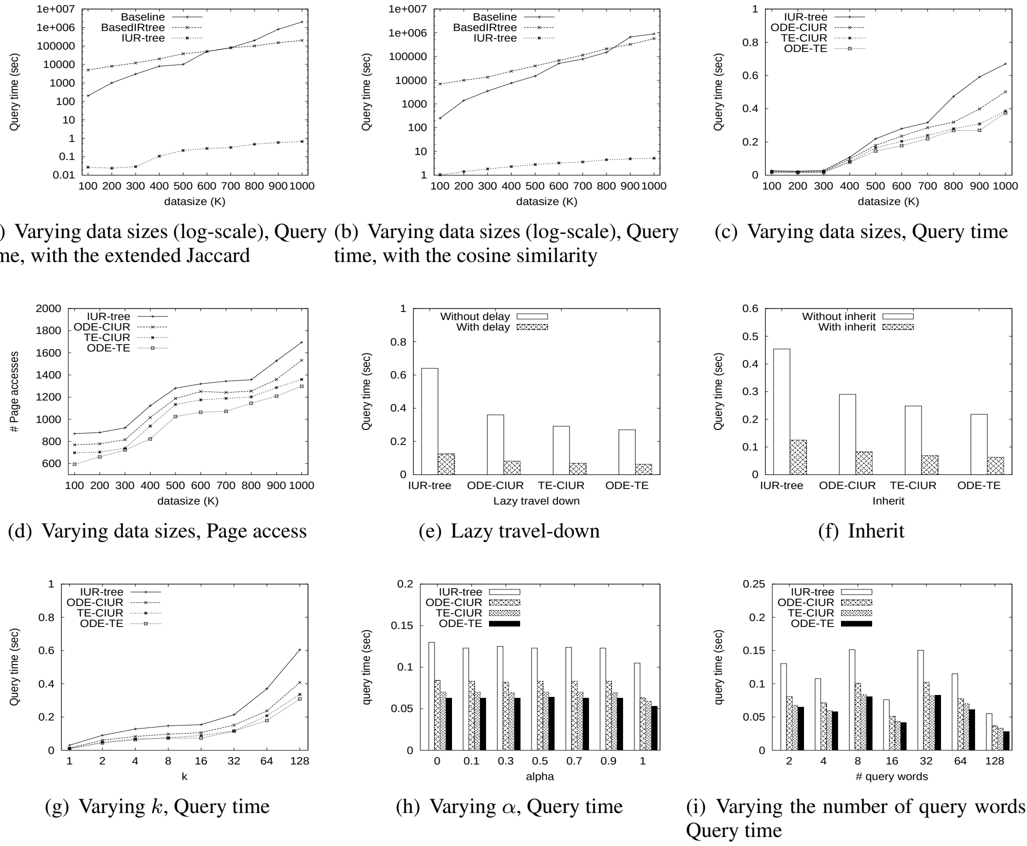
8.1. Experiments for search algorithms on IUR-trees and CIUR-trees

In the first set of experiments, we studied the performance and the scalability of different algorithms on IUR-trees and CIUR-trees when varying data sizes. In particular, we generated different datasets ranging from 100K to 1000K by randomly selecting objects from the original GN dataset shown in Table III. As baselines, we implemented two methods described in Section 4, which are based on the threshold algorithm (TA) [Fagin et al. 2003] (called *Basline*) and the IR-trees [Cong et al. 2009] (called *BasedIRTree*), respectively. Furthermore, we also studied the performances of algorithms based on the cosine similarity for textual similarity measurement.

8.1.1. Baselines vs. IUR-trees. Fig. 14(a) exhibits the running time of three algorithms based on two baselines and the IUR-tree. Note that the query time is shown in log scale. The IUR-tree clearly outperforms two baselines by orders of magnitude. The gap becomes larger when the size of datasets gets bigger, which empirically verifies the theoretical analysis about the computation complexity of baselines and our algorithms in Section 6.3. In addition, *BasedIRtree* outperforms *Baseline* when the size of data is larger than 600K. It is because *BasedIRtree* uses the IR-tree to find k spatial-textual neighbors and its superiority over the TA algorithm (in *Baseline*) becomes clear only in the setting of a large data set.

Fig. 14(b) shows the performance of various algorithms based on the cosine similarity. Again, the IUR-tree is clearly the winner in this setting and performs significantly better than two baselines.

Comparing Figures 14(a) and 14(b), we have some additional insights: the performances of *Baseline* based on the extended Jaccard and the cosine similarity are similar, whereas the performances of *BasedIRtree* and IUR-tree based on the cosine similarity perform worse than those based on the extended Jaccard. The reason is that the lower and upper bounds (in Eqn. 19) between two index nodes for the cosine similarity are not as tight as those for the extended Jaccard. In addition, *BasedIRtree* and IUR-tree rely on the bounds of textual similarity to prune nodes, but $Baseline$ does not use those bounds.



(a) Varying data sizes (log-scale), Query time, with the extended Jaccard

(b) Varying data sizes (log-scale), Query time, with the cosine similarity

(c) Varying data sizes, Query time

(d) Varying data sizes, Page access

(e) Lazy travel-down

(f) Inherit

(g) Varying k, Query time

(h) Varying α, Query time

(i) Varying the number of query words, Query time

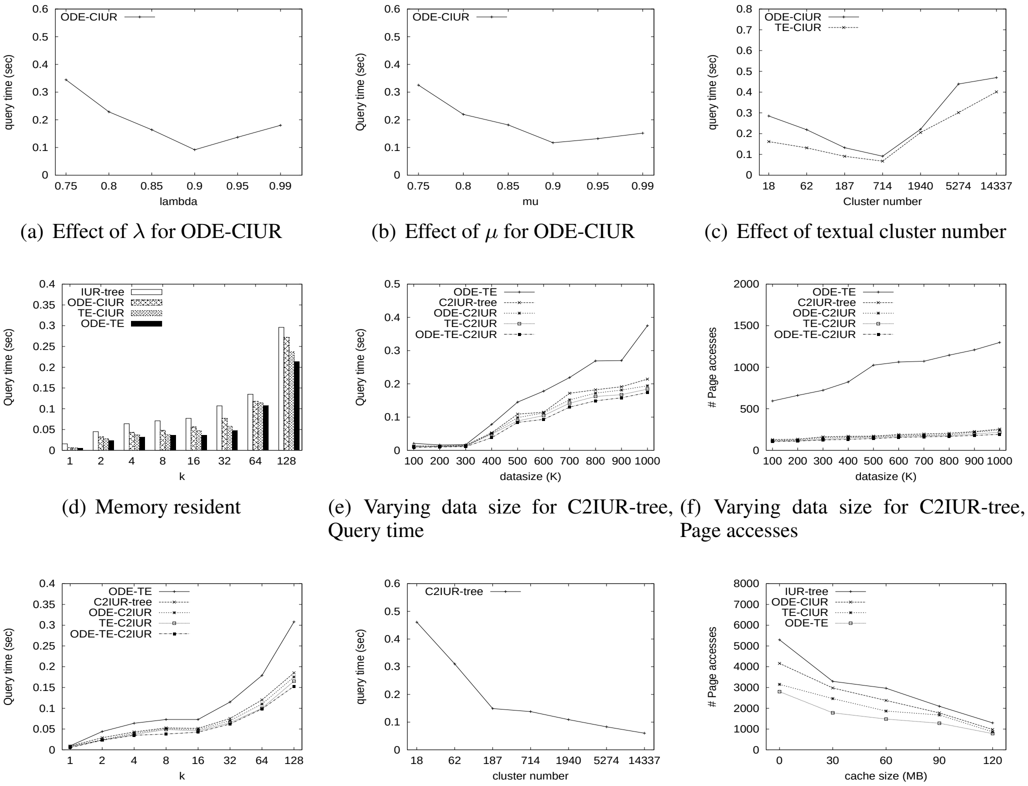
Fig. 14. Experimental results on the GN dataset

8.1.2. IUR-trees vs. CIUR-trees. In Fig. 14(c) and Fig. 14(d), we observe that the CIUR-tree-based algorithms outperform the IUR-tree-based approaches, which indicates that the two optimizations (i.e. Outline Detection and Extraction (ODE) and Text Entropy (TE) optimizations) enhance the filtering power and reduce the number of index nodes visited. Note that the ODE-TE algorithm that combines the two optimization approaches is the fastest algorithm in this experiment and scales well with the size of the data sets.

8.1.3. Effect of "lazy travel-down" and "inherit". To demonstrate the usefulness of the techniques used in our RSK*k*NN algorithm: "*lazy travel-down*" and "*inherit*" individually, we study the performance when one of the two techniques is turned off. First, Figure 14(e) shows that the "*lazy travel-down*" approach speeds up all four algorithms by more than 50%. This is because it can avoid visiting some irrelevant entries. Second, Figure 14(f) shows the benefit of *inherit* technology for all the algorithms. We observe that *inherit* can significantly improve the performance since it avoids computing contribution lists from the scratch.

8.1.4. Effect of parameters k, α and qw. In this set of experiments, we study how system performance is affected by the following parameters: the number of returned top results k, the combination ratio of the similarity function α and the number of words in queries qw. The results are reported in Figures 14(g)$\sim$ 14(i).

Figures 14(g) shows the runtime *w.r.t.* k. We fix α=0.7 and qw=16, and vary k from 1 to 9. The results show that the runtime and required I/O of our algorithms increase slightly as k grows.



(a) Effect of λ for ODE-CIUR (b) Effect of μ for ODE-CIUR (c) Effect of textual cluster number

(d) Memory resident (e) Varying data size for C2IUR-tree, Query time (f) Varying data size for C2IUR-tree, Page accesses

(g) Varying k for C2IUR-tree (h) Varying cluster number for C2IUR-tree (i) Speedup vs cache size, Page accesses

Fig. 15. Experimental results on the GN dataset

To evaluate the impact of α, we vary α from 0 to 1, thus adjusting the importance between textual similarity and spatial proximity. Figure 14(h) shows that our algorithm is insensitive to α. At α=1, i.e., text documents are totally ignored, the runtime is obviously shorter, as expected.

Figures 14(i) shows the results when we vary the number of query words qw. We can see that the algorithms run faster as we increase qw from 2 to 32. This is because more query words may improve the pruning power by decreasing the average textual similarity between query words and data points.

8.1.5. Effect of parameters λ and μ for ODE-CIUR. This experiment is to study the effects of parameters λ and μ for ODE-CIUR. As shown in Fig. 15(a), with the increase of λ, the runtime first decreases and then increases. The interplay between λ and the running time can be illustrated as follows. Smaller λ means that the condition of Case I (described in Section 7.3) is easier to be satisfied and thus more entries are identified as Case I, which will lead to unnecessary accesses of entries. However, if the value of λ is too large (e.g., 0.99), the condition of Case I is too strict to be satisfied and thus the benefit of the optimization is marginal. Intuitively, there is a sweet spot between the two extremes. In our experiments, we found the best spot to be λ=0.9. As shown in Fig. 15(b), the effect of parameter μ is similar to that of the parameter λ. When μ is around 0.9, the performance is the best.

8.1.6. Effect of cluster number. As shown in Fig. 15(c), the performance of both ODE-CIUR and TE-CIUR is insensitive to the number of clusters, and they achieve the best performance when the number of cluster is around 200-1000. We can see that the runtime decreases as the number of clusters changes from 18 to 187, but increases as the number changes from 1,940 to 14,337. This is because the time needed for processing clusters counteracts the time saved by the text cluster enhancement.

8.1.7. Memory-resident implementation. This experiment is to evaluate the performance of the algorithms on memory-resident indexes. Since the size of the two ranking lists in the baseline method is too big to fit in memory, we evaluate the performance of the other four algorithms on the GN dataset when the indexes are in memory. As shown in Fig. 15(d), when varying the parameter k, ODE-TE outperforms the other algorithms, which is consistent with the results reported on disk-resident indexes.

8.1.8. Effect of cache. This experiment is to evaluate the impact of the cache strategy on our algorithms. We used the well-known LRU cache method [Johnson and Shasha 1994], and varied the cache size from 0 to 120MB, where 120MB corresponds to about 20% of IUR-tree. As shown in Fig. 15(i) on the GN dataset, the cache method improves the I/O performance of all the algorithms. This is expected since caches save the I/O cost by reducing page accesses. Note that the cache strategy does not change the trend in the performance of different methods. That is the ODE-TE CIUR-tree, which combines two optimization strategies, is still the best of all the five algorithms.

8.2. Experiments for search algorithms on C^{2}IUR-tree

8.2.1. Performance of different algorithms and scalability. In this set of experiments, we evaluate the performance and the scalability of various algorithms on the C^{2}IUR-tree. As shown in Figures 15(e) and 15(f), all the search algorithms based on the C^{2}IUR-tree (including the C^{2}IUR, ODE-C^{2}IUR, TE-C^{2}IUR and ODE-TE-C^{2}IUR) outperform the ODE-TE algorithm which is the fastest among the search algorithms based on the CIUR-tree and its optimizations (ODE-CIUR, TE-CIUR and ODE-TE). Therefore, all the search algorithms based on the C^{2}IUR-tree are superior to the algorithms on the CIUR-tree and its optimizations. In addition, Figures 15(e) and 15(f) show that the optimization algorithms “Outlier Detection and Extraction” and “ Textual Entropy”, are also applicable to the C^{2}IUR-tree to prune the irrelevant objects and improve the performance. Moreover, we observe that the ODE-TE-C^{2}IUR algorithm that combines the two optimization approaches based on the C^{2}IUR-tree is the fastest among all the algorithms in our experiments.

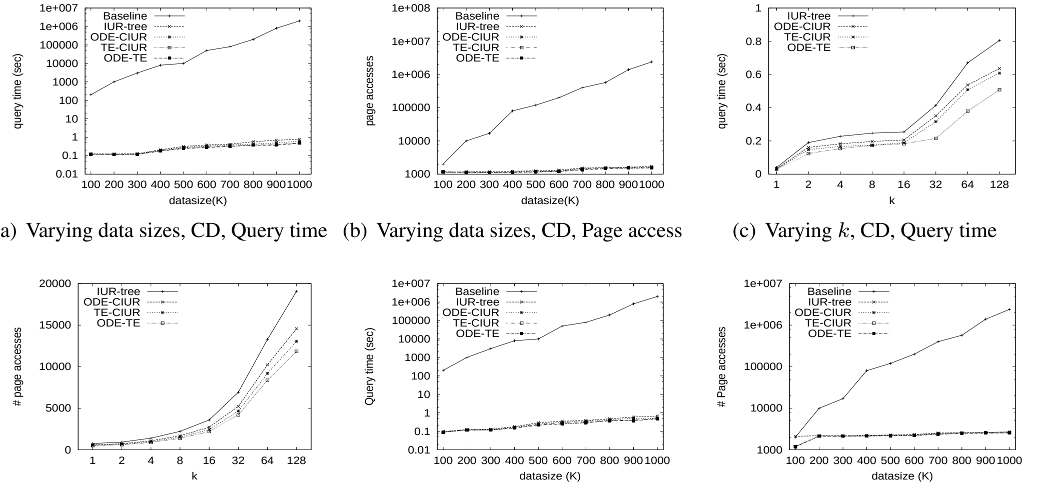
8.2.2. Effect of parameter k. This experiment is to evaluate the impact of parameter k on the performance of the search algorithms on the C^{2}IUR-tree. As shown in Figure 15(g), by varying parameter k from 1 to 128, the runtime of our algorithms increase slightly with the increase of k.

8.2.3. Effect of cluster number. As discussed in Section 7.2, we use textual clusters to compute the textual entropy in the construction of C^{2}IUR-trees. In this set of experiments, we evaluated the effect of the number of clusters on the performance of the C^{2}IUR-tree. Note that this experiment is different from the one evaluating the effect of the number of clusters on the search optimizations based on the CIUR-tree in Figure 15(c) in Section 8.1.6. Figure 15(h) demonstrates that the query time tends to decline with the increase of the number of clusters. The reason is that with a larger number of clusters, the textual entropy would become more precise to represent the textual information in an index node, thus improving the efficiency of the algorithms.

8.3. Experiments on other datasets

All the above experimental results are reported on the GN dataset. We also conducted extensive sets of experiments on the other two datasets Shop and CD, and part of the experimental results are shown in Figures 16(a)-16(f). We can see that the trends of both sets of experimental results are consistent with those of the GN dataset. Therefore, they demonstrate the effectiveness and efficiency of our algorithms under different datasets with various data-sizes, spatial-distributions, word-cardinalities and text-sizes.

To summarize, our experimental results show that the proposed hybrid indexes and search algorithms outperform the baseline method, and the two optimizations with text-entropy and outline-detection based on CIUR-tree and C^{2}IUR-tree can further improve the performance of the RSKkNN query.



(a) Varying data sizes, CD, Query time (b) Varying data sizes, CD, Page access (c) Varying k, CD, Query time

(d) Varying k, CD, Page access (e) Varying data sizes, Shop, Query time (f) Varying data sizes, Shop, Page access

Fig. 16. Experimental results on CD and Shop datasets

9. Conclusions and Future Work

In this paper we introduced and addressed a new problem called RSKkNN queries, which is an extension of RkNN queries where the search criteria is based on the fusion of spatial information and textual description. This extension renders the existing solutions to answer RkNN queries inapplicable to RSKkNN queries. Thus, we presented the IUR-tree and its two optimizations CIUR-tree and C^{2}IUR-tree to represent and index the hybrid information and proposed the RSKkNN algorithm, which quickly computes contribution lists, and adjusts the thresholds to prune unrelated points and identify true hits as early as possible. We also provided a new cost model to theoretically analyze the performance of our algorithms. Finally, we conducted extensive experiments to verify the scalability and the performances of our proposed algorithms and optimizations.

As for the future work, this article opens a number of promising directions. First, we plan to extend our algorithms to the bichromatic version of RSKkNN queries, considering the textual relevance for documents belonging to two different types of objects. Second, we intend to consider other variants of RSKkNN queries, such as the skyline RSKkNN queries. Finally, we would like to develop algorithms for the scenarios where the spatial objects are moving, uncertain objects or objects that are constrained to a road network.

REFERENCES

ACHTERT, E., BÖHM, C., KRÖGER, P., AND KUNATH, P. 2006. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *SIGMOD*. 515–526.

ACHTERT, E., KRIEGEL, H.-P., KRÖGER, P., RENZ, M., AND ZÜFLE, A. 2009. Reverse k-nearest neighbor search in dynamic and general metric databases. In *EDBT*. 886–897.

A.FOX, E., CHEN, Q. F., M.DAOUD, A., AND S.HEATH, L. 1991. Order-preserving minimal perfect hash functions and information retrieval. In *TOIS*. 281–308.

BERCHTOLD, S., BÖHM, C., KEIM, D., AND KRIEGEL, H. 1997. A cost model for nearest neighbour search in high-dimensional data space. In *Proceedings 16th ACM Conference on Principles of Database Systems(PODS)*. 78–86.

BOHM, C. AND KRIEGEL, H. 2001. A cost model and index architecture for the similarity join. In *Proceedings 17th IEEE International Conference on Data Engineering (ICDE)*. 411–420.

BORIAH, S., CHANDOLA, V., AND KUMAR, V. 2008. Similarity measures for categorical data: A comparative evaluation. In *SDM*. 243–254.

CAO, X., CONG, G., AND JENSEN, C. S. 2010. Retrieving top-k prestige-based relevant spatial web objects. *PVLDB 3,* 1, 373–384.

CHEEMA, M. A., LIN, X., ZHANG, W., AND ZHANG, Y. 2011. Influence zone: Efficiently processing reverse k nearest neighbors queries. In *ICDE*. 577–588.

CHEEMA, M. A., LIN, X., ZHANG, W., AND ZHANG, Y. 2012. Efficiently processing snapshot and continuous reverse k nearest neighbors queries. In *Proceedings of the VLDB Journal*.

CHEEMA, M. A., LIN, X., ZHANG, Y., 0011, W. W., AND ZHANG, W. 2009. Lazy updates: An efficient technique to continuously monitoring reverse knn. *PVLDB 2,* 1, 1138–1149.

CONG, G., S.JENSEN, C., AND WU, D. 2009. Efficient retrieval of the top-k most relevant spatial web objects. In *PVLDB*. 337–348.

CORRAL, A., MANOLOPOULOS, Y., THEODORIDIS, Y., AND VASSILAKOPOULOS, M. 2006. Cost models for distance joins queries using r-trees. In *Data & Knowledge Engineering*. 1–36.

DEGROOT, M. H. AND SCHERVISH, M. J. 2004. Probability and statistics. In *Pearson Education (US)*.

EMRICH, T., KRIEGEL, H.-P., KROGER, P., RENZ, M., XU, N., AND ZUFLE, A. 2010. Reverse k-nearest neighbor monitoring on mobile objects. In *GIS*. 494–497.

ESTER, M., KRIEGEL, H.-P., SANDER, J., AND XU, X. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*. 226–231.

FAGIN, R., LOTEM, A., AND NAOR, M. 2003. Optimal aggregation algorithms for middleware. In *J.Comput. Syst. Sci.* 614–656.

FALOUTSOS, C. AND KAMEL, I. 1994. Beyond uniformity and independence: Analysis of r-trees using the concept of fractal dimension. In *Proceeding of the 13th ACM SIGACT-SIGMODE-SIGART Symposium on Principles of Database Systems*. 4–13.

FALOUTSOS, C., SELLIS, T. K., AND ROUSSOPOULOS, N. 1987. Analysis of object oriented spatial access methods. In *SIGMOD Conference*. 426–439.

GUTTMAN, A. 1984. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*. 47–57.

HAVELIWALA, T. H., GIONIS, A., KLEIN, D., AND INDYK, P. 2002. Evaluating strategies for similarity search on the web. In *Proceedings of the 11th international conference on World Wide Web*. WWW '02. 432–442.

HUANG, A. 2008. Similarity measures for text document clustering. In *In New Zealand Computer Science Research Student Conference*. 49–56.

HUANG, Y., JING, N., AND E.A.RUNDENSTEINER. 1997. A cost model for estimating the performance of spatial joins using r-trees. In *Proceedings 9th International Conference on Scientific and Statistical Database Management (SSDBM)*. 30–38.

I.D.FELIPE, V.HRISTIDIS, AND N.RISHE. 2008. Keyword search on spatial databases. In *ICDE*. 656–665.

JOHNSON, T. AND SHASHA, D. 1994. 2q: A low overhead high performance buffer management replacement algorithm. In *VLDB*. 439–450.

KAMEL, I. AND FALOUTSOS, C. 1993. On packing r-trees. In *ICDE*. 490–499.

KANG, J. M., MOKBEL, M. F., SHEKHAR, S., XIA, T., AND ZHANG, D. 2007. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *ICDE*. 806–815.

KHODAEI, A., SHAHABI, C., AND LI, C. 2012. Skif-p: a point-based indexing and ranking of web documents for spatial-keyword search. *Geoinformatica 16,* 3, 563–596.

KORN, F. AND MUTHUKRISHNAN, S. 2000. Influenced sets based on reverse nearest neighbor queries. In *SIGMOD*. 201–212.

KORN, F., PAGEL, B., AND FALOUTSOS, C. 2001. On the 'dimensionlity curse' and the 'self-similarity blessing'. In *IEEE Transactions on Knowledge and Data Engineering*. 96–111.

KULLBACK, S. AND LEIBLER, R. A. 1951. On information and sufficiency. *The Annals of Mathematical Statistics 22,* 1, 79–86.

LEE, M. D. AND WELSH, M. 2005. An empirical evaluation of models of text document similarity. In *In CogSci2005*. 1254–1259.

LI, Z., LEE, K. C. K., ZHENG, B., LEE, W.-C., LEE, D. L., AND WANG, X. 2011. Ir-tree: An efficient index for geographic document search. *IEEE Trans. Knowl. Data Eng. 23,* 4, 585–599.

LIN, K.-I., NOLEN, M., AND YANG, C. 2003. Applying bulk insertion techniques for dynamic reverse nearest neighbor problems. In *IDEAS*. 290–297.

LU, J., LU, Y., AND CONG, G. 2011. Reverse spatial and textual k nearest neighbor search. In *SIGMOD Conference*. 349–360.

N.ROUSSOPOULOS, S.KELLEY, AND F.VINCENT. 1995. Nearest neighbor queries. In *SIGMOD*. 71–79.

PAGEL, B., SIX, H.-W., TOBEN, H., AND WIDMAYER, P. 1993. Towards an analysis of range query performance in spatial data structures. In *Proceeding of the 12th ACM SIGACT-SIGMODE-SIGART Symposium on Principles of Database Systems*. 214–221.

PAPADOPOULOS, A. AND MANOLOPOULOS, Y. 1997. Performance of nearest neighbour queries in r-trees. In *Proceeding of the 6th International Conference on Database Theory*. 394–408.

SALTENIS, S., JENSEN, C. S., LEUTENEGGER, S. T., AND LOPEZ, M. A. 2000. Indexing the positions of continuously moving objects. In *Proceedings of the ACM SIGMOD Conference*. 331–342.

SALTON. 1988. Term-weighting approaches in automatic text retrieval. In *Information Processing and Management*. 513–523.

SINGH, A., FERHATOSMANOGLU, H., AND TOSUN, A. S. 2003. High dimensional reverse nearest neighbor queries. In *CIKM*. 91–98.

STANOI, I., AGRAWAL, D., AND ABBADI, A. E. 2000. Reverse nearest neighbor queries for dynamic databases. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. 44–53.

STANOI, I., RIEDEWALD, M., AGRAWAL, D., AND ABBADI, A. 2001. Discovery of influence sets in frequently updated databases. In *VLDB*. 99–108.

STREHL, A., STREHL, E., GHOSH, J., AND MOONEY, R. 2000. Impact of similarity measures on web-page clustering. In *In Workshop on Artificial Intelligence for Web Search (AAAI 2000)*. AAAI, 58–64.

TAN, P.-N., STEINBACH, M., AND KUMAR, V. 2005. *Introduction to Data Mining*. Addison-Wesley.

TAO, Y. AND PAPADIAS, D. 2003. Spatial queries in dynamic environments. In *ACM Transactions on Database Systems (TODS)*. Vol. 28.

TAO, Y., PAPADIAS, D., AND LIAN, X. 2004. Reverse knn search in arbitrary dimensionality. In *VLDB*. 744–755.

TAO, Y., ZHANG, J., PAPADIAS, D., AND MAMOULIS, N. 2004. An efficient cost model for optimization of nearest neighbour search in low and medium dimensional spaces. In *IEEE Transactions on Knowledge and Data Engineering*. 1169–1184.

THEODORIDIS, Y. AND SELLIS, T. 1996. A model for the prediction of r-tree performance. In *PODS*. 161–171.

THEODORIDIS, Y., STEFANAKIS, E., AND SELLIS, T. 2000. Efficient cost models for spatial queries using r-trees. In *IEEE Transactions on Knowledge and Data Engineering*. 19–32.

TITCHMARSH, E. C. 2005. The theory of the riemann zeta-function. In *Oxford University Press*.

VAID, S., JONES, C. B., JOHO, H., AND SANDERSON, M. 2005. Spatio-textual indexing for geographical search on the web. In *SSTD*. 218–235.

VLACHOU, A., DOULKERIDIS, C., KOTIDIS, Y., AND NØRVÅG, K. 2010. Reverse top-k queries. In *ICDE*. 365–376.

WU, W., YANG, F., CHAN, C. Y., AND TAN, K.-L. 2008a. Continuous reverse k-nearest-neighbor monitoring. In *MDM*. Beijing, 132–139.

WU, W., YANG, F., CHAN, C.-Y., AND TAN, K.-L. 2008b. Finch:evaluating reverse k-nearest-neighbor queries on location data. In *PVLDB*. 1056–1067.

ZHANG, D., CHEE, Y. M., MONDAL, A., TUNG, A. K. H., AND KITSUREGAWA, M. 2009. Keyword search in spatial databases: Towards searching by document. In *ICDE*. 688–699.

ZHOU, Y., XIE, X., WANG, C., GONG, Y., AND MA, W.-Y. 2005. Hybrid index structures for location-based web search. In *Proceedings of the 14th ACM international conference on Information and knowledge management*. CIKM '05. ACM, New York, NY, USA, 155–162.

Received June 2012; revised June 2013; accepted January 2014