Reverse Spatial and Textual k Nearest Neighbor Search

Jiaheng Lu Ying Lu
DEKE, MOE and School of Information
Renmin University of China
{jiahenglu, yinglu}@ruc.edu.cn

Gao Cong
School of Computer Engineering
Nanyang Technological University, Singapore
gaocong@ntu.edu.sg

ABSTRACT

Geographic objects associated with descriptive texts are becoming prevalent. This gives prominence to spatial keyword queries that take into account both the locations and textual descriptions of content. 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 paper, we define Reverse Spatial Textual k Nearest Neighbor (RSTkNN) query, i.e., finding objects that take the query object as one of their k most spatial-textual similar objects. Existing works on reverse kNN queries focus solely on spatial locations but ignore text relevance.

To answer RSTkNN queries efficiently, we propose a hybrid index tree called IUR-tree (Intersection-Union R-Tree) that effectively combines location proximity with textual similarity. Based on the IUR-tree, we design a branch-and-bound search algorithm. To further accelerate the query processing, we propose an enhanced variant of the IUR-tree called clustered IUR-tree and two corresponding optimization algorithms. Empirical studies show that the proposed algorithms offer scalability and are capable of excellent performance.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

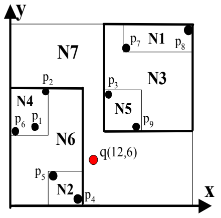
Reverse k nearest neighbor, Spatial-keyword query

1. INTRODUCTION

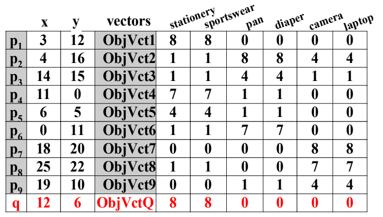
Reverse k Nearest Neighbor (RkNN) [10] query, which is to find objects whose k nearest neighbors (kNN) include the query point, has received considerable attention. Among many applications, it is used to discover the influence sets, *i.e.*, objects in a dataset highly influenced by the query object [10]. In the literature [2, 3, 7, 10, 19, 20, 22, 25], spatial distance is usually considered as the sole influence factor. However, in real applications, distance alone is not sufficient to characterize the influence between two objects. For example, two objects, *e.g.*, restaurant, are more likely to influence each other if their textual descriptions (*e.g.*, seafood buffet lunch including crab and shrimp) are similar.

In contrast, we take into account textual similarity in RkNN, and study a new kind of RkNN problem that is called **Reverse Spatial and Textual k Nearest Neighbor (RSTkNN)** queries that consider both spatial distance and textual similarity. An RSTkNN query is to find the objects that have the query object as one of their k most spatial-textual similar objects. This new type of query is different from RkNN (e.g., [10]), and spatial-keyword queries (*e.g.*, LkT [6]). (See Section 8 for a detailed comparison).

Figure 1 gives an example to illustrate the RSTkNN query we proposed and the conventional RkNN query. Points $p_1 \cdots p_9$ in Fig.1(a) are existing branch stores in a region, and q is a new store which will open. (N1$\cdots$N7 in Fig.1(a) are MBRs to be explained in Section 4). Products of each branch store are given in Fig.1(b), where the weight of each item can be calculated by TF-IDF [18]. Then an RSTkNN query with q as query object finds the existing stores that will be influenced most by q considering both the locations of the stores and the stuffs that the stores sell. Assume k=2, the results of 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 is one of answers since the textual description of p_1 is quite similar with that of q, though q is not a 2NN of p_1 in terms of spatial distance alone.



(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)

Figure 1: An example of RSTkNN queries

RSTkNN queries have many applications ranging from map-based Web search to GIS decision support. For example, a shopping mall can use an RSTkNN query 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 want to buy/rent a house would describe her/his desired house with both location and textual description that specifies the amenities (s)he wants. RSTkNN 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 RSTkNN will pose great challenges to the existing techniques for processing conventional RkNN (without considering textual relevance), and render them inapplicable to process RSTkNN queries (see Section 3.1 for the detailed analysis).

To process the RSTkNN queries efficiently, we propose a hybrid indexing structures and an efficient approach that take into account the fusion of location proximity and document similarity. The contributions and the organization of this paper are summarized as follows.

1. We propose and analyze the problem of Reverse Spatial and Textual k Nearest Neighbor (RSTkNN) search. To the best of our knowledge, this is the first work on RSTkNN queries (Section 2, 3).

2. We propose an efficient algorithm to process RSTkNN queries. The algorithm is based on an effective hybrid indexing structure called Intersection-Union-R tree (IUR-tree) that stores spatial and textual information for the database objects (Section 4). By effectively computing spatial-textual similarities between index nodes, we exploit a branch-and-bound algorithm to prune the irrelevant subtrees (Section 5). We also theoretically analyze the performance of the algorithm based on IUR-tree.

3. IUR-trees organize the data points by considering only spatial distance, which may damp the pruning power due to the diversity of textual contents in one node. we propose an enhanced hybrid index, called Clustered IUR-tree (CIUR-tree) incorporating textual clusters and two optimization algorithms based on CIUR-tree (Section 6).

4. Results of empirical studies with implementations of the proposed techniques demonstrate the scalability and efficiency of proposed indexes and algorithms (Section 7).

2. PROBLEM DEFINITION

In this work, the document of an object is treated as a bag of weighted words using vector space model [14]. 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 could be computed by the well-known TF-IDF scheme [18].

Let P be a universal spatial object set. Each spatial 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 RSTkNN query as follows. Given a set of objects P and a query point q (loc,vct), RSTkNN(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 previous work [6,12], 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 spatial proximity factor and the textual similarity factor. Note that in our setting, we allow users to adjust the parameter α at the query time.

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

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

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

As shown in Eqn(2), the spatial distance $SimS(.,.)$ of objects p_1, $p_2 \in P$ is the Euclidean distance denoted as $dist(p_1.loc, p_2.loc)$. 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 [21], which is widely used in textual similarity computing, as shown in Eqn(4).

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

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

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

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

$$RSTkNN(q,k,P) = \{p \in P | q \in STkNN(p,k,P)\} \quad (5)$$

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 $RSTkNN(q,k,P)$=$\{p_1, p_4, p_5, p_9\}$. Note that p_1 is an answer due to the high textual similarity between p_1 and q.

3. PROBLEM ANALYSIS AND BASELINE METHODS

We first analyze the main technical challenges of processing RSTkNN queries and then present the baseline solutions.

3.1 Problem Analysis

Recall that our similarity metric combines both text and location information, and to the best of our knowledge, the previous methods for RkNN queries cannot be employed to handle the RSTkNN queries due to the new challenges.

Existing approaches for processing RkNN can be grouped into four categories. First, typical solutions (e.g. [3,13,15,22,25]) are based on ℓp norm metric space. For example, hyperplanes [15,22,25] and Voronoi cells [13] technologies are employed to exploit the Euclidean geometry properties; and [3] makes use of the properties of ℓp norm metric to develop a branch-and-bound algorithm to answer RkNN queries. Despite their significant contributions, the ℓp norm metric space is not suitable for computing the similarity

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

between textual descriptions as they are sparse and high-dimensional vectors [21], and hence the techniques based on the Euclidean geometry are not applicable to RSTkNN queries. Second, the approach to prune the objects using the hyper-Voronoi diagram and 60-degree-pruning method [20] is infeasible to process RSTkNN due to the high dimensions of text. Third, the existing methods based on pre-processing [2,7,10] do *not* suffice the requirement of RSTkNN that the importance of text relevance verse spatial proximity (controlled by parameter α) may vary at query time. In addition, they cannot handle dynamic parameter k, which may vary at querying time too. Fourth, the works [1,19] focus on only approximate results while they are successful to handle high dimensional and general metric spaces.

3.2 Baseline methods

No baseline algorithm exists for *RSTkNN* queries. Readers might be tempted to answer *RSTkNN* queries by separately computing the reverse spatial k nearest neighbors (*RSkNN*) and the reverse textual k nearest neighbors(*RTkNN*), and then finding a way to combine the two results. This idea has two serious problems: 1) existing methods cannot be used to compute RkNN queries in terms of text similarity, and more importantly 2) there is no sensible way to combine the two results to get the answers since the result of an $RSTkNN$ query may even not belong to the set of the union of the results from its corresponding $RSkNN$ query and $RTkNN$ query.

We develop a non-trivial baseline algorithm to correctly find *all* answers. The idea is that: for each object p, we first pre-compute the location proximity and textual similarity respectively with *all* the other objects to obtain two sorted lists. At query time, we find top-k objects that have the highest overall grades of the function in Eqn(1) combining the two lists using the threshold algorithm (TA) [9]. Thus if the spatial-textual similarity of the k-th object is smaller than the similarity between p and query q, then p is added to the result. This method can handle dynamic parameters k and α. However, it is expensive as it demands computing the $STkNN$ for *all* the objects in the dataset.

We also explore other baselines using existing techniques. However, they are much slower than the above baseline in our experiments, and are ignored. For example, we tried a baseline method without pre-computation. The idea is that, for each object p, find its top-k most similar objects using the existing spatial-and-textual *kNN* query techniques (*e.g.*, [6]), and if the kth result is smaller than the similarity between p and query q, then p is added to the result.

4. A HYBRID INDEX: IUR-TREE

To answer the RSTkNN queries efficiently, we propose an effective hybrid index called IUR-tree (Intersection-Union R-tree), which is a combination of textual vectors and R-tree [11] We can also use other R-tree-based indexes to build IUR-tree. In particular, each node of an IUR-tree contains both spatial location and textual information. The former is represented with a minimum bounding rectangle(MBR), and the latter is represented with two textual vectors: an intersection vector and a union vector. The two vectors are used to compute the similarity approximations (Section 5.1.1 shows the formulas later).

Leaf nodes contain 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 the intersection and union textual vectors that intersect and union all textual vectors in the entries of the child node, respectively. 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. Note that the two vectors are stored in a separate storage region; and 4)cnt is the number of objects (in the leaf nodes) in the subtree rooted at Ptr.

Figure 2 illustrates the IUR-tree for the objects in Figure 1. The intersection and union textual vectors are presented in Fig.3. 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 $N1$.

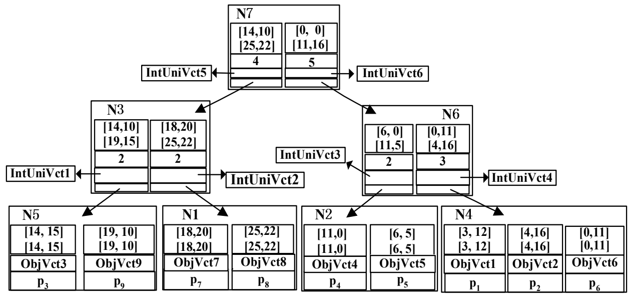
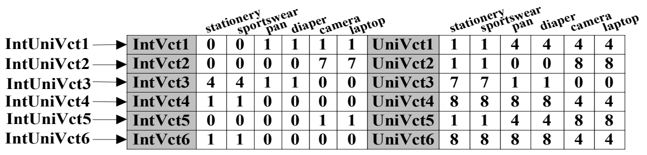


Figure 2: The IUR-tree of Figure 1



		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

Figure 3: Text vectors for IUR-tree in Figure 2

Algorithm 1 is presented to construct the IUR-tree, which is using an insert operation that is adapted from the corresponding R-tree operation [11]. To update an IUR-tree incrementally, we use *order preserving minimal perfect hashing function* (*OPMPHF*) [4] to store 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, $d_i.p$ is an integer (position in the words collection) hashed from word d_i using *OPMPHF*, and $d_i.w$ is the weight of word d_i. In particular, in Algorithm 1, Function $Convert()$ in Line 1 is to convert a document to a vector in the form of ($d_i.p$, $d_i.w$). Line 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 (Line 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.

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

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

```
1: $TextVct$←Convert($document$); //Covert document into text vector in form of $(d_i.p, d_i.w)$.
2: $N$ ← ChooseLeaf($MBR$);
3: add $TextVct$ and $MBR$ to node $N$;
4: if $N$ needs to be split then
5:     {$O$, $P$} ← $N$.split();
6:     if $N$.isroot() then
7:         initialize a new node $M$;
8:         $M$.append($O.MBR$, $O.TextVct$);
9:         $M$.append($P.MBR$, $P.TextVct$);
10:        StoreNode($M$);
11:        StoreNode($O$);
12:        StoreNode($P$);
13:        $R$.RootNode ← $M$;
14:    else
15:        AdjustTree($N.ParentNode$, $O$, $P$);
16: else
17:    StoreNode($N$);
18:    if ¬$N$.isroot() then
19:        AdjustTree($N.ParentNode$, $N$, null);
```

5. RSTKNN QUERY ALGORITHM

We present a novel approach to compute the lower and upper bounds of similarity between a node in the IUR-tree and its kth most similar objects in Section 5.1. Based on the bounds, we present a branch-and-bound algorithm to answer RSTkNN queries in Section 5.2.

5.1 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 objects, denoted by $kNN^L(E)$ and $kNN^U(E)$, respectively.

5.1.1 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. Since the spatial distance approximation is thoroughly studied in several publications concerning spatial indexing (*e.g.*, [3,17]), here we do not repeat their detailed definitions. Rather, we concentrate on the more specialized textual part. The proofs of lemmas in this section can be found in Appendix A.

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 1 ($MinST$). *The minimal spatial-textual similarity between two entries E and E′ in IUR-tree, denoted by $MinST(E,E')$, is defined as:*

$$MinST(E,E') = \alpha(1 - \frac{MaxS(E,E') - \varphi_s}{\psi_s - \varphi_s}) + (1-\alpha)\frac{MinT(E,E') - \varphi_t}{\psi_t - \varphi_t} \tag{6}$$

where $MaxS(E,E')$ is the maximal Euclidian distance between two MBRs of E and E′, and

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

$$\begin{cases} E.w_j = E.u_j, E'.w_j = E'i_j & \text{if } E.i_j * E.u_j \geq E'.i_j * E'.u_j \\ E.w_j = E.i_j, E'.w_j = E'.u_j & \text{otherwise} \end{cases} \tag{7}$$

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

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 2 ($TightMinST$). *A tight minimal spatial-textual similarity between two entries E and E′ in IUR-tree, denoted as $TightMinST(E,E')$, is defined as:*

$$TightMinST(E,E') = max\left\{ \alpha(1 - \frac{MinMaxS(E,E') - \varphi_s}{\psi_s - \varphi_s}) + (1-\alpha)\frac{MinT(E,E') - \varphi_t}{\psi_t - \varphi_t}, \alpha(1 - \frac{MaxS(E,E') - \varphi_s}{\psi_s - \varphi_s}) + (1-\alpha)\frac{TightMinT(E,E') - \varphi_t}{\psi_t - \varphi_t} \right\} \tag{8}$$

where, $MinMaxS(E,E')$ [3] is a tighter Euclidian distance function than $MaxS(E,E')$ holding the property that there is at least one object o′ in E′ such that the distances between o′ and all the objects in E are smaller or equal to $MinMaxS(E,E')$. $TightMinT(E,E') =$

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

$$E'.w_r = \begin{cases} E'.u_r & \text{if } E.i_r * E.u_r > E'.i_r * E'.u_r \\ E'.i_r & \text{otherwise} \end{cases} \tag{9}$$

$$E.w_r = \begin{cases} E.i_r & \text{if } \sqrt{E.i_r * E.u_r} < E'.w_r; \\ E.u_r & \text{otherwise}; \end{cases} \tag{10}$$

and $E.w_j$ and $E'.w_j$ are assigned as Eqn(7).

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

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 3 ($MaxST$). *The maximal spatial-textual similarity between two entries E and E′ in IUR-tree, denoted as $MaxST(E,E')$, is defined as:*

$$MaxST(E,E') = \alpha(1 - \frac{MinS(E,E') - \varphi_s}{\psi_s - \varphi_s}) + (1-\alpha)\frac{MaxT(E,E') - \varphi_t}{\psi_t - \varphi_t} \tag{11}$$

where $MinS(E,E')$ is the minimal Euclidian distance between two MBRs of E and E′; maximal textual similarity between E and E′ $MaxT(E,E') = \frac{\sum_{j=1}^{n} E.w_j \times E'.w_j}{\sum_{j=1}^{n} E.w_j^2 + \sum_{j=1}^{n} E'.w_j{}^2 - \sum_{j=1}^{n} E.w_j \times E'.w_j}$,

$$\begin{cases} E.w_j = E.i_j,\ E'.w_j = E'.u_j & \text{if } E.i_j > E'.u_j \\ E.w_j = E.u_j,\ E'.w_j = E'.i_j & \text{if } E.u_j < E'.i_j \\ E.w_j = E'.w_j = E.u_j & \text{if } E'.i_j \leq E.u_j \leq E'.u_j \\ E.w_j = E'.w_j = E'.u_j & \text{otherwise.} \end{cases} \tag{12}$$

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

Corollary 1. *There is at most one object o′ in E′ s.t. $\forall o{\in}E$, $SimST(o,o'){=}MaxST(E,E')$.*

5.1.2 *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 4 (Lower bound contribution list). *Let $\mathcal{T}$ be a set of entries in the IUR-tree that do not have ancestor-descendant relationship. Given an entry $E \in \mathcal{T}$, a lower bound contribution list of E, denoted as $E.^LCL$, 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 $E'_i \neq E$, $E' \in \mathcal{T}$, s_i is $MinST(E, E')$ or $TightMinST(E, E')$, and*

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

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

For $s_i = MinST(E, E')$, the rationale for subtracting one from $|E'|$ is due to the potential presence of one object in E' with more precise approximation by $TightMinST$. Following the notations in Definition 4, 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).*

The proof of Lemma 4 is obvious: when the condition holds, there are at least k distinct objects whose similarities with any object $o \in E$ are larger than or equal to $MaxST(E, q)$. 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 5 (Upper bound contribution list). *Let $\mathcal{T}$ be a set of entries in the IUR-tree that do not have ancestor-descendant relationships. An upper bound contribution list of an entry E in IUR-tree, denoted as $E.^UCL$, is a sequence of t pairs $<s_i, E'_i>$, $i \in [1, \cdots, t]$, sorted in descending order of s_i where $E'_i \neq E$, $E' \in \mathcal{T}$, $s_i = MaxST(E, E')$, and t is the maximal number fulfilling $1 + \sum_{i=1}^{t-1} |E'_i| \leq k$.*

Following the notations in Definition 5, 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.*

The proof of Lemma 5 is obvious: there are at most k-1 distinct objects whose similarities with any object $o \in E$ are equal to or larger than $MinST(E, q)$, and thus all objects in E will be reported as part of the answer. Thus we let the upper bound $kNN^U(E)$ be $E.^UCL.s_t$. That is we can report E to be a result entry if $kNN^U(E) < MinST(E, q)$.

Figure 4 illustrates the strategies of using $kNN^L(E)$ and $kNN^U(E)$ to determine whether 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

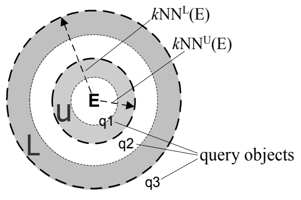


Figure 4: Illustration of pruning and reporting results using $kNN^L(E)$ and $kNN^U(E)$

$q3$ 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.*, is 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 $q1$ is within the ring $kNN^U(E)$. If query is $q2$, we cannot determine whether E belongs to results based on $kNN^L(E)$ and $kNN^U(E)$.

5.2 Search algorithm

We proceed to develop an efficient algorithm to answer RSTkNN queries (see Algorithm 2). 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 list; 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 objects 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* that will not be results, and a result object *ROL*.

The algorithm begins with initialization and then enqueues the root of the IUR-tree into U (Line 1–2). When U is not empty (Line 3), we dequeue the entry P from U with the highest priority (Line 4). 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), based on which, we determine whether E is a result entry ("*hit*") or can be pruned ("*drop*") by invoking procedure IsHitOrDrop (Line 7). If E can be pruned (Line 29), E is added to *PEL*, and if E is reported as a result entry (Line 33), E is added to *ROL*; Otherwise, we use E to "*mutual-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 and 12). If E' is pruned or reported as a result entry then remove E' from its original data structure U or *COL* (Line 13–14). If E is determined as a hit or drop, then consider next child entry of P (Line 10). If E still cannot be determined whether to be a result entry after effected by all the entries in *COL*, *ROL* and U, then add E to the corresponding list or queue (Line 15–17). Finally, when the priority queue U is empty, we still need to process objects in the candidate list *COL* to decide if they belong to answers by invoking Procedure FinalVerification (Line 18).

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, 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.

Algorithm 2 RSTkNN (R: IUR-tree root, q: query)

```
Output: All objects $o$, s.t. $o \in RSTkNN(q,k,R)$.
 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 $COL$, $ROL$, $U$ 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($COL$, $PEL$, $ROL$);

    Procedure FinalVerification($COL$, $PEL$, $q$)
20: while ($COL \neq \emptyset$) do
21:   Let $E$ be an entry in $PEL$ with the lowest level;
22:   $PEL$=$PEL$-$\{E\}$;
23:   for each object $o$ in $COL$ do
24:     UpdateCL($o$,$E$);//update contribution lists of $o$.
25:     if (IsHitOrDrop($o$, $q$)=true) then
26:       $COL$=$COL$-$\{o\}$;
27:   for each child entry $E'$ of $E$ do
28:     $PEL$=$PEL \cup \{E'\}$; //access the children of $E'$

    Procedure IsHitOrDrop($E$: entry, $q$: query)
29: if $kNN^L(E) \geq MaxST(E,q)$ then
30:   $PEL$.append($E$); //Lemma 4
31:   return true;
32: else
33:   if $kNN^U(E) < MinST(E,q)$ and $E$ is the rightest child
      entry then
34:     $ROL$.append($subtree(E)$); //Lemma 5
35:     return true;
36:   else return false;

    Procedure UpdateCL($E$: entry, $E'$: entry)
37: for each tuple $<s_i, E'_i, num_i> \in E.^LCL$ do
38:   if $E'_i = E$ or $E'_i$=Parent($E$) then
39:     remove $<s_i, E'_i, num_i>$ from $E.^LCL$; //Clean Con-
        flicts
40: if $kNN^U(E) < MaxST(E,E')$ then
41:   $E.^UCL \leftarrow$ TopkMax($E.^UCL$, $MaxST(E,E')$, 1);
42: if $kNN^L(E) < TightMinST(E,E')$ then
43:   $E.^LCL \leftarrow$ TopkMax($E.^LCL$, $TightMinST(E,E')$, 1);
44: if $kNN^L(E) < MinST(E,E')$ then
45:   $E.^LCL \leftarrow$ TopkMax($E.^LCL$, $MinST(E,E')$, $|E'|$-1);

    SubProcedure TopkMax($L$,$f(E,E')$,$C$)
46: Return the $t$-th triple in contribution list $L$, where $t$ is
    the minimal number fulfilling $\sum_{i=1}^{t} L.num_i \geq k$.
```

Procedure FinalVerification: 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 candidate in *COL* until we can correctly determine if each candidate object belongs to an answer or not. In particular, Line 20 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 23 uses the entry E to update the contribution list of each candidate o in *COL* and Line 24 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 26–27). This process continues until *COL* becomes empty.

In particular, Line 6 in Algorithm 2 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 37–38 in Algorithm 2 guarantee that there is no object in contribution lists which is double counted, as illustrated in the following example.

Table 1: Trace of *RSTkNN* algorithm in Example 1

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,p3
7	Verify p9	∅	∅	p4,p1,p5,p9	N1,p3

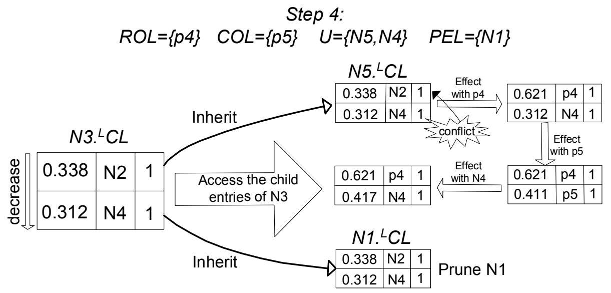


Figure 5: Illustration to *RSTkNN* algorithm

Example 1: *We use this example to illustrate RSTkNN 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 Fig. 4). After $N3$ is dequeued from U in Step 4, we access its child entries $N5$ and $N1$.

1) $N5$ also inherits the contribution list from $N3$ (Line 6 in Algorithm 2). However, it can neither be pruned nor be determined to be results (Line 7). Thus it will "mutual-effect" with p4, p5, and $N4$ (Line 8–17). When we consider *the effect of* $p4$ *on the contribution list* $(N5.^{L}CL)$ *of* $N5$ *(Line 9),* $N2$ *(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 38), 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 16).*

2) $N1$ *inherits the contribution list from* $N3$ *(Line 6) and is pruned immediately according to Lemma 4 (Line 29) 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 2 correctly returns all RSTkNN points.*

The proof of Theorem 1 is given in Appendix A.4.

Performance Analysis: We propose an analytical model to estimate the cost of *RSTkNN* queries and theoretically analyze the performance of the *RSTkNN* algorithm based on IUR-tree, *i.e.*. We estimate the number of expected disk accesses DA of IUR-tree index nodes by the *RSTkNN* algorithm as follows.

LEMMA 6. *Suppose that* N *objects are uniformly distributed in 2-dimension space [23], and* M *distinct words are randomly distributed on the* N *objects such that the average number of words per object is* m $(m \leq M)$*. The number of expected disk accesses* DA *is* $\mathcal{O}(k\sqrt{k} + \sqrt{k}N)$*, where* f *is the average capacity (fanout) of the IUR-tree,* k *is the parameter specified by the RSTkNN query.*

The details for the analysis model and the proof of Lemma 6 can be found in [16].

6. 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 6.1. We present two optimization methods to improve the search performance based on CIUR-tree in Section 6.2 and Section 6.3, respectively.

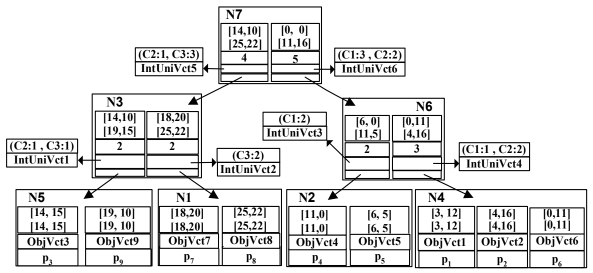
6.1 Cluster IUR-tree: CIUR-tree

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.

The intersection and union vectors at each node of the IUR-tree is also extended with cluster information. 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 6(a). The CIUR-tree is shown in Fig 6(b).

		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

Figure 6: The Cluster IUR-tree of Figure 1

6.2 Outlier Detection and Extraction

To give a tighter bound during CIUR-tree traversal, we need to purify the textual description in the index nodes by extracting outliers from the documents based on cluster IDs. We take the outliers into special account and calculate their bounds separately so that the bounds of the original entry can be tighter. We identify the index node E containing outlier clusters in the following two cases:

Case *I*: Most objects in *subtree*(E) can be pruned, but there exit 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 exits 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 exit 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 exits 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 decompose them and identify if their subtree entries can be pruned or added as results. To implement the optimization of outlier detection and extraction for *RSTkNN* queries, the only change is to replace Line 17 in Algorithm 2 with the following pseudocodes. First, we determine whether the index node E is in Case I or II: if not, then enter E into priority queue U; if yes, then we check whether E's subtree entries e 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 2

```
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($U$, $E$);
```

6.3 Text-entropy based optimization

We proceed to propose the second optimization based on CIUR-tree to improve performance. In particular, we use $TextEntropy$ to depict the distribution of text clusters in an entry of CIUR-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 in CIUR-tree 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 into 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 optimizations in Section 6.2 and Section 6.3 is that they are orthogonal and can be combined, as implemented in our experiments.

7. EXPERIMENTAL STUDIES

We present an experimental study to evaluate the efficiency and scalability of our methods to answer *RSTkNN* queries.

Implemented algorithms. We implemented the proposed algorithm based on IUR-tree, the two optimizations based on CIUR-tree: outlier-detection-extraction optimization (ODE-CIUR) and text-entropy optimization (TE-CIUR), and the combination of two optimizations (ODE-TE). In addition, we also implemented the baseline method (Section 3.2). The other baseline discussed in Section 3.2 performs much worse and is ignored here.

Table 2: Datasets for the experiments

Statistics	Shop	CD	GN
total # of objects	304,008	1,555,209	1,868,821
total unique words in dataset	3933	21,578	222,409
average # words per object	45	47	4

Datasets and Queries. The algorithms were evaluated using three datasets: ShopBranches (Shop), GeographicNames (GN), and CaliforniaDBpedia (CD). 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 usefulness and efficiency of our algorithms in different environments. The statistics of each dataset are shown in Table 2. 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 document collection of abstracts about California in DBpedia (wiki.dbpedia.org/Downloads351). Finally, the ShopBranches are extended from a small real data describing 955 shop branches and their products by generating new spatial objects while maintaining the distribution of the real location and textual information of the objects.

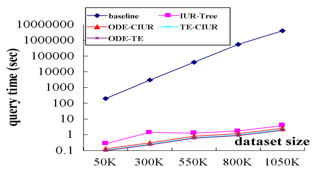
For each dataset, we generated 7 sets of query sets, in which the number of words is 2, 4, 8, 16, 32, 64 and 128, respectively. Each query set comprises 100 queries that are randomly selected from the respective dataset. We report the average running time of 100 queries for each query set.

Setup and metrics. We implemented all the algorithms with VC++6.0, and an Intel(R) Core(TM)2 Quad CPU Q8200 @2.33GHz with 4GB of RAM. We implemented algorithms based on both disk-resident and memory-resident for the proposed index structures, namely the IUR-tree and the CIUR-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 set at 0.9 by default. In CIUR-tree, we clustered the textual vectors of objects into different number of clusters using the *DBSCAN* [8] cluster algorithm.

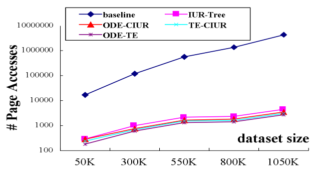
We compared varied algorithms with different experimental settings as follows:

parameter k:	*1 ∼ 9,*	*default 3*
parameter α:	*0 ∼ 1,*	*default 0.7*
number of query words qw:	*1∼128,*	*default 16*
number of clusters:	*18∼14337,*	*default 187*

Space requirement. The space requirement for the structures used by the algorithms is shown in Table 3. Obviously, the baseline method needs the largest disk space as it needs to store the two lists of objects ranked by location proximity and textual similarity respectively for each object. The disk requirement of the baseline on CD and GN is beyond the capacity of our computer (800G). As for the disk storage requirement of IUR-tree, compared with the original R-tree [11], the number of non-leaf nodes are comparable with that of the R-tree and the size of text vectors in the entries

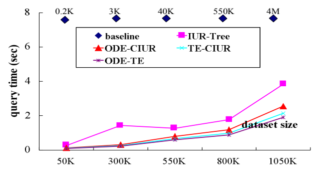


(a) Query time

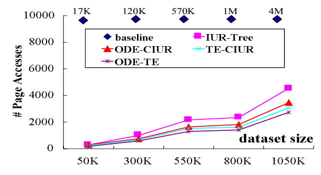


(b) Page access

Figure 7: Varying dataset sizes for Shop (log-scale)

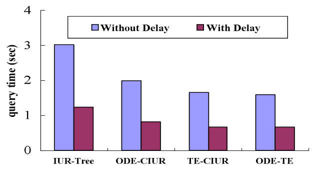


(a) Query time

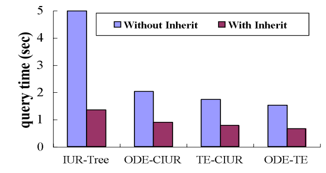


(b) Page access

Figure 8: Varying dataset sizes for Shop

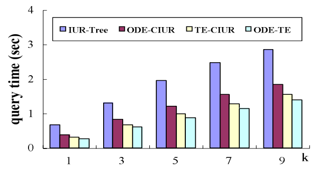


(a) Lazy Travel Down

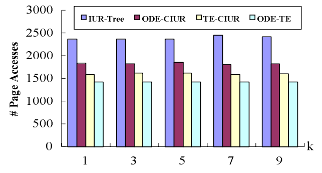


(b) Inherit

Figure 9: Study on the Proposed Algorithm

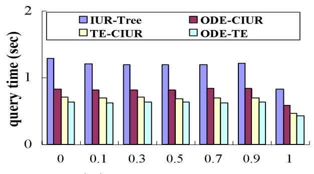


(a) Query time

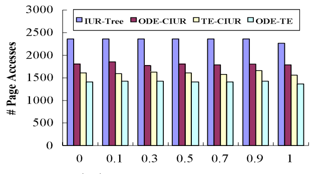


(b) Page access

Figure 10: Varying k for Shop

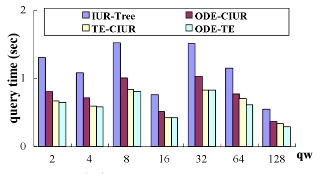


(a) Query time

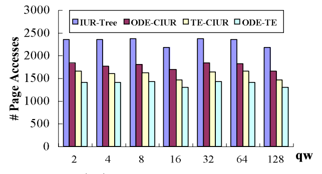


(b) Page access

Figure 11: Varying α for Shop

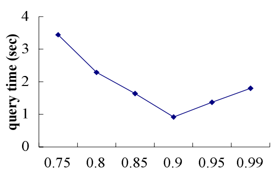


(a) Query time

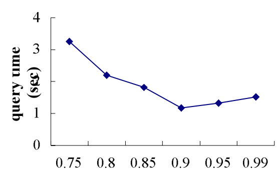


(b) Page access

Figure 12: Varying the number of query words



(a) λ



(b) μ

Figure 13: Effect of λ and μ for ODE-CIUR

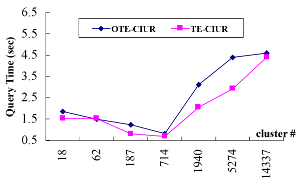


Figure 14: Effect of textual cluster number

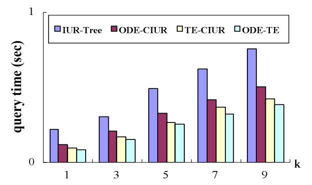


Figure 15: Memory Resident for Shop

of leaf nodes of IUR-tree is comparable with the size of the text components of the dataset. Thus the additional disk requirement for IUR-tree is the text vectors in the entries of non-leaf nodes. For each entry, the size of its intersection vector in an index node is at most the maximal number of words in an object, and the size of its union vector is at most the number of distinct words in the dataset. The space of CIUR-tree is a bit larger than IUR-tree since it needs extra space to store the intersection and union vectors for each cluster.

Table 3: Sizes of indexing structures for Shop

Data	Two lists for Baseline	IUR-tree	CIUR-tree
Shop	224GB	40MB	47MB
CD	-	218MB	237MB
GN	-	264MB	306MB

Performance of different algorithms and scalability In the first set of experiments, we evaluated the performances and the scalability of various algorithms using the ShopBranches dataset. As shown in Fig. 7 (note the log-scale), our proposed algorithms significantly outperform the baseline method for datasets of different sizes in terms of query time and the number of node accesses by orders of magnitude. This is primarily because the baseline method computes the $STkNN$ for *all* the objects in the dataset (Note that the $SkNN$ and $TkNN$ of each object has been pre-computed and stored in disk).

In Fig. 8, we replotted the same data as Fig. 7 using linear scale to see the differences of various optimizations clearly. We observed that the $CIUR$-tree-based algorithms outperform IUR-tree-based approaches, which indicates that the two optimizations enhance the filtering power and reduce the number of index nodes visited. Note that the ODE-TE algorithm that combines two optimized approaches is the fastest algorithm in our experiments and scales well with the sizes of data sets.

To demonstrate the usefulness of the techniques used in our $RSTkNN$ algorithm: "*lazy travel-down*" and "*inherit*", we made experiments and plotted Fig. 9(a) and 9(b). (1) The "*lazy travel-down*" approach speeds up all four algorithms by over 50%. This is because it can avoid visiting some irrelevant entries. (2) Fig.9(b) shows the benefit of *inherit* technology for all the algorithms. We can see that *inherit* can significantly improve the performance since it could avoid the computation of contribution lists from the scratch.

Effect of parameters k, α and qw. In the second set of experiments, we sought to analyze how system performance is impacted by three 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 10∼ 12.

(1) Figure 10 shows the runtime and number of page accesses $w.r.t$ 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 with the increase of k.

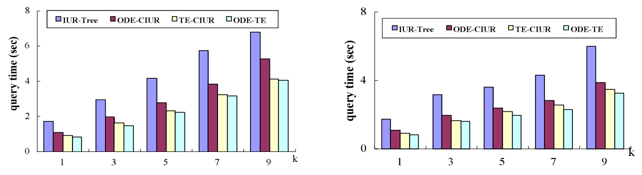
(2) To evaluate the impact of α, we vary α from 0 to 1, thus adjusting the importance between textual similarity and spatial proximity. From Fig. 11, we can see that our algorithm is not sensitive to α. When α=1 meaning that text documents are totally ignored, the runtime is obviously shorter, as expected.

(3) Figure 12 shows the results when we vary the number of query words qw. We can see that algorithms run faster with the increase of qw. In particular, while qw is varied from 2 to 32, the more the query words, the faster the algorithms run. A deep analysis indicates that more query words may improve the pruning power by decreasing the average textual similarity between query words and data points.

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.13(a), with the increase of λ, the runtime first decreases then increases since most entries would be treated as Case I to be traveled down if λ is too small and most entries cannot identified as Case I to lose the benefits of the optimization of ODE-CIUR if λ is too large. An appropriate setting of λ is about 0.9 in our experiment. The effect of parameter μ shown in Fig.13(b). When μ is about 0.9, the performance is the best in our experiment.

Effect of cluster number. As shown in Fig.14, the performance of both ODE-CIUR and TE-CIUR is not sensitive to the number of clusters, and they achieve the best performance when the number of cluster is around 200-1000. We can see that runtime decreases from 18 clusters to 187 clusters, but then increases from 1,940 clusters to 14,337 clusters. This behavior occurs because the time needed for processing clusters counteracts the time saved by text cluster enhancement.

Memory-resident implementation. This experiment is to evaluate the performance of algorithms on memory-resident. Since the baseline method cannot be fit in memory due to the large size of two ranking lists, we evaluated the performance of the other four algorithms in memory using the dataset ShopBranches. As shown in Fig. 15, when varying the parameter k, the performance of all the algorithms speed up at least 50% since there is no I/O cost for the indexes comparing Fig. 10 for disk resident.



(a) Varying k for CD (b) Varying k for GN

Figure 16: Results of varying k for data CD and GN

Experiments on other datasets. All the above experimental results are reported based on ShopBranches data set. We also conducted extensive experiments on the other two datasets GeographicNames and CaliforniaDBpedia. Due to the space limitation, we only give their results for varying the parameter k in RSTkNN in Fig.16(a) and Fig.16(b) respectively. Both of the experimental results are consistent with those on dataset ShopBranches. The baseline method does not work on the two datasets since the disk space required by preprocessing is beyond the limitation of disk capacity.

To summarize, our experimental results show that our proposed hybrid indexes and search algorithms outperform the baseline method. We also show that the two optimizations with text-entropy and outline-detection indeed improve the performance of the *RSTkNN* queries.

8. RELATED WORK

Our *RSTkNN* queries combine reverse k nearest neighbor (RkNN) queries and textual similarity searches. Existing works [7, 10, 15, 22, 25] on RkNN queries mainly focus on spatial proximity. As discussed in Section 3.1, the existing approaches are not applicable to answer RSTkNN queries. Recent work *Reverse top-k queries* [24] retrieves RkNN objects in a weighted feature space, which, however, does not consider spatial proximity. Therefore existing technologies on RkNN queries cannot be applied on *RSTkNN* queries.

Recently, queries on spatial objects associated with textual information have received significant attentions. *Top*-k *spatial keyword queries* [12], *Location-aware top-k text retrieval(LkT) query* [6], *m-closest keywords (mCK) query* [26], and *prestige-based spatial-keyword query* [5], combine keyword queries with spatial queries. However, none of them considers reverse kNN query. Additionally, the hybrid index structures in this paper is different from the hybrid index structures used in the existing work on spatial-keyword query. One main difference is that we augment R-tree with intersection and union text vectors to represent the text component so that we can compute/estimate the textual similarity between objects/entries, while the existing hybrid indexes usually augment R-tree nodes with inverted file or signature file to compute the relevance between objects/entries and query. In particular, the IR^2-tree [12] integrating the R-tree and signature, and the bR*-tree [26] combining the R*-tree with bitmap are used for Boolean keyword query, but not IR-style ranking queries. They are quite different from the IUR-tree. The IR-tree [6] augments a node with inverted lists, which are suitable for keyword queries that just concern about the query keywords; while in the IUR-tree a node is augmented with text vector, which is suitable for similarity search. Moreover, IR-tree keeps only union information of text while the IUR-tree contain union and intersection information, which are required to compute the similarity approximations between entries. Clustering is also used to optimize query processing in the IR-tree [6], where the clustering is used in a different manner. Hence, the proposed algorithms in this paper are very different from the algorithms for answering spatial-keyword query.

9. CONCLUSIONS AND FUTURE WORK

This paper addressed the new problem $RSTkNN$ query, which is the extension of $RkNN$ query with the fusion of spatial information and textual description, making it much richer and more complex for the construction and traverse of the index. This paper presented the IUR-tree to represent and index the hybrid information and proposed $RSTkNN$ algorithms to quickly compute contribution lists, and adjust the thresholds to prune unrelated points and identify true hits as early as possible. As for the future works, this paper opens to a number of promising directions. First, it is necessary to extend our algorithms to bichromatic version, considering the semantical relevancy for documents of two types of objects. Second, it would be interesting to consider some variants of *RSTkNN* queries, such as skyline *RSTkNN* queries. Third, it is of interest to develop algorithms if the spatial objects are moving objects, fuzzy objects, or are constrained to a road network.

10. ACKNOWLEDGEMENTS

This paper benefitted tremendously from many insightful comments from Tok Wang Ling, Wei Wu, and Xiaofang Zhou as well as the anonymous reviewers. Jiaheng Lu was sponsored partially by the NSF China (60903056) and Beijing Municipal Natural Science Foundation (4112030).

11. REFERENCES

[1] E. Achtert, C. Böhm, P. Kröger, and P. Kunath. Approximate reverse k-nearest neighbor search in general metric spaces. In *CIKM*, pages 788–789, 2006.

[2] E. Achtert, C. Böhm, P. Kröger, and P. Kunath. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *SIGMOD*, pages 515–526, 2006.

[3] E. Achtert, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle. Reverse k-nearest neighbor search in dynamic and general metric databases. In *EDBT*, pages 886–897, 2009.

[4] E. A.Fox, Q. F. Chen, A. M.Daoud, and L. S.Heath. Order-preserving minimal perfect hash functions and information retrieval. In *TOIS*, pages 281–308, 1991.

[5] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *PVLDB*, 3(1):373–384, 2010.

[6] G. Cong, C. S.Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. In *PVLDB*, pages 337–348, 2009.

[7] C.Yang and K.I.Lin. An index structure for efficient reverse nearest neighbor queries. In *ICDE*, pages 485–492, 2001.

[8] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.

[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *J.Comput. Syst. Sci.*, pages 614–656, 2003.

[10] F.Korn and S.Muthukrishnan. Influenced sets based on reverse nearest neighbor queries. In *SIGMOD*, pages 201–212, 2000.

[11] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[12] I.D.Felipe, V.Hristidis, and N.Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665, 2008.

[13] I.Stanoi, M.Riedewald, D.Agrawal, and A.E.Abbadi. Discovery of influence sets in frequently updated databases. In *VLDB*, pages 99–108, 2001.

[14] K.Nigam, A.K.McCallum, S.Thrun, and T.M.Mitchel. Text classification from labeled and unlabeled documents using em. In *Machine Learning*, pages 103–134, 2000.

[15] H. P. Kriegel, P. Kröger, M. Renz, A. Züfle, and A. Katzdobler. Incremental reverse nearest neighbor ranking. In *ICDE*, pages 1560–1567, 2009.

[16] J. Lu, Y. Lu, and G. Cong. Technical report, renmin university of china. 2011.

[17] N.Roussopoulos, S.Kelley, and F.Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.

[18] Salton. Term-weighting approaches in automatic text retrieval. In *Information Processing and Management*, pages 513–523, 1988.

[19] A. Singh, H. Ferhatosmanoglu, and A. S. Tosun. High dimensional reverse nearest neighbor queries. In *CIKM*, pages 91–98, 2003.

[20] I. Stanoi, D. Agrawal, and A. E. Abbadi. Reverse nearest neighbor queries for dynamic databases. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 44–53, 2000.

[21] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[22] Y. Tao, D. Papadias, and X. Lian. Reverse knn search in arbitrary dimensionality. In *VLDB*, pages 744–755, 2004.

[23] Y. Theodoridis and T. Sellis. A model for the prediction of r-tree performance. In *PODS*, pages 161–171, 1996.

[24] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Reverse top-k queries. In *ICDE*, pages 365–376, 2010.

[25] W. Wu, F. Yang, C.-Y. Chan, and K.-L. Tan. Finch:evaluating reverse k-nearest-neighbor queries on location data. In *PVLDB*, pages 1056–1067, 2008.

[26] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699, 2009.

APPENDIX

A. PROOFS

A.1 Proof of Lemma 1

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

DEFINITION 6 (SIMILARITY PRESERVING FUNCTION). *Given two functions $fsim$: $V \times V \rightarrow R$ and $fdim$: $R \times R \rightarrow R$, where V denotes the domain of n-element vectors and 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 2. *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 2 is true.

CLAIM 3. *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 (13) is true by means of mathematical induction. Thus extend 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 3 holds.

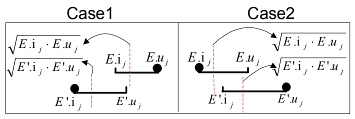
$$\frac{\sum_{i=1}^{n} x_i x'_i}{\sum_{i=1}^{n} \frac{x_i^2 + {x'_i}^2}{2}} \geq \frac{\sum_{i=1}^{n} x_i x''_i}{\sum_{i=1}^{n} \frac{x_i^2 + {x''_i}^2}{2}} \tag{13}$$

$$\Rightarrow \frac{\sum\limits_{i=1}^{n} x_i x'_i}{\sum\limits_{i=1}^{n} x_i^2 + \sum\limits_{i=1}^{n} {x'_i}^2 - \sum\limits_{i=1}^{n} x_i x'_i} \geq \frac{\sum\limits_{i=1}^{n} x_i x''_i}{\sum\limits_{i=1}^{n} x_i^2 + \sum\limits_{i=1}^{n} {x''_i}^2 - \sum\limits_{i=1}^{n} x_i x''_i}$$

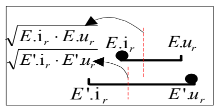
$$\Rightarrow \quad EJ(\overrightarrow{p}, \overrightarrow{p'}) \geq EJ(\overrightarrow{p}, \overrightarrow{p''})$$

Based on Claim 3, we proceed to prove the property of $MinST$, which is the fusion of $MaxS$ and $MinT$.

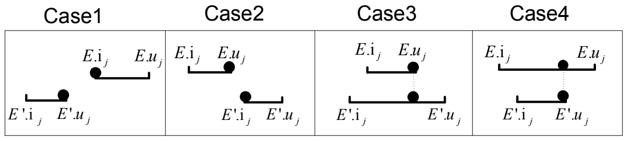
$MinT$ in Eqn(7): For each dimension j, as shown in Fig.17(a), when $\sqrt{E.i_j \cdot E.u_j} \geq \sqrt{E'.i_j \cdot E'.u_j}$ (Case 1), *i.e.*,



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



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



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

Figure 17: Assignments of approximations

$\frac{E'.i_j}{E.u_j} \leq \frac{E.i_j}{E'.u_j}$, 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 3, 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(6), 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(6) can guarantee that $\forall o \in E$, $\forall o' \in E'$, $MinST(E, E') \leq SimST(o, o')$, *i.e.*, Lemma 1 is true. □

A.2 Proof of Lemma 2

PROOF. Eqn(8) suggests that $TightMinST$ is composed of $MinMaxS$, $MinT$, $MaxS$ and $TightMinT$, which have the following properties respectively.

$MinMaxS$: According to the work [3], $MinMaxS$ has the property that there exists an object $o \in E'$, so that $\forall o' \in E$, $dist(o, o') \leq MinMaxS(E, E')$.

$MaxT$ in Eqn(12): As shown in Fig.17(c), for each dimension j, if $E.i_j > E'.u_j$ (Case 1), then $\forall E.w \in [E.i_j, E.u_j]$ and $\forall E'.w \in [E'.i_j, E'.u_j]$, we have $\frac{E'.u_j}{E.i_j} \geq \frac{min\{E.w, E'.w\}}{max\{E.w, E'.w\}}$, thus based on Claim 3, the assignment $E.w_j = E.i_j$, $E'.w_j = E'.u_j$ can guarantee that $\forall$ $o \in$ $subtree(E)$, $\forall o' \in$ $subtree(E')$, $MaxT(E, E') \geq SimT(o, o')$. It is similar if $E.u_j < E'.i_j$ (Case 2). For other cases, *i.e.*, the intervals $[E.i_j, E.u_j]$ and $[E'.i_j, E'.u_j]$ overlap, so obviously, $E.w_j$ and $E'.w_j$ should be assigned as the same value according to the preserving function. Further, both of $E.w_j$ and $E'.w_j$ should be assigned the largest value in the intersection between $[E.i_j, E.u_j]$ and $[E'.i_j, E'.u_j]$ shown in Case 3 and Case 4 since Extended Jaccard function is an increasing function as the increase of the values. Thus the assignment in Eqn(12) satisfies the property of $MaxT(E, E')$.

$TightMinT$ in Eqn(10): As shown the assignment of one dimension r in Fig.17(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 Fig.17(a). Therefore, according to Claim 3, 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(8): 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 $\exists o' \in E'$, $\forall o \in E$, $SimST(o, o') =$ α(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. □

A.3 Proof of Lemma 3

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

A.4 Proof of Theorem 1

PROOF. (Sketch) We prove that (1) Algorithm *RSTkNN* is correct, that is all returned objects are desired answers; and that (2) the returned results are complete.

Correctness: The search strategy in *RSTkNN* algorithm is to prune entries E in the tree using the lower bound of spatial-textual *kNN* of E: *kNN*$^L(E)$ (Line 29) and to report entries E using the upper bound *kNN*$^U(E)$ (Line 33). *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, we can prove the correctness of the technique of "inherit" (Line 6) and "lazy travel-down" (Line 8) based on the observation that the similarity approximations of ancestor entries are more conservative than that of descendant entries.

Completeness: All objects which can not be safely pruned or reported as results, are appended to the candidate object list *COL* (Line 17). 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 24) while $MinST$ and $MaxST$ between two database objects are equal. Thus our algorithm is complete, *i.e.*, it can return all the *RSTkNN* data points.

Hence, Theorem 1 is true. □