Maximizing Bichromatic Reverse Spatial and Textual k Nearest Neighbor Queries

Farhana M. Choudhury J. Shane Culpepper Timos Sellis
School of CSIT, RMIT University, Melbourne, Australia
{farhana.choudhury, shane.culpepper, timos.sellis}@rmit.edu.au

Xin Cao
School of EEE and CS, Queen's University, Belfast, UK
x.cao@qub.ac.uk

ABSTRACT

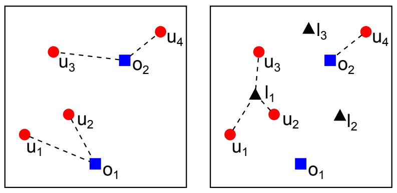
The problem of maximizing bichromatic reverse k nearest neighbor queries (BRkNN) has been extensively studied in spatial databases. In this work, we present a related query for spatial-textual databases that finds an optimal location, and a set of keywords that maximizes the size of bichromatic reverse spatial textual k nearest neighbors (MaxBRSTkNN). Such a query has many practical applications including social media advertisements where a limited number of relevant advertisements are displayed to each user. The problem is to find the location and the text contents to include in an advertisement so that it will be displayed to the maximum number of users. The increasing availability of spatial-textual collections allows us to answer these queries for both spatial proximity and textual similarity. This paper is the first to consider the MaxBRSTkNN query. We show that the problem is NP-hard and present both approximate and exact solutions.

1. INTRODUCTION

Bichromatic reverse k nearest neighbor (BRkNN) [10, 17] and maximizing BRkNN (MaxBRkNN) queries [21, 20, 26, 12, 11] have received considerable attention in the spatial database community. Given two sets of spatial objects O and P over a shared dataspace, a BRkNN query issued by an object $p \in P$ returns all the objects $o \in O$ that have p as a kNN in P. A MaxBRkNN query finds the optimal region to place an object p such that the cardinality of the results of the BRkNN query issued by p in O is maximized.

A practical application of BRkNN queries is to find potential customers who would visit a restaurant based on their locations. Given the same context, a MaxBRkNN query would find a region to establish a new restaurant p such that p is a kNN of the maximum number of customers. In the literature, spatial distance is usually the sole relevance factor. However, customers are generally interested in the products and services as well as the location. Therefore, spatial-textual queries are of interest in these settings.

Top-k spatial-textual queries [3, 25, 16] return the k most similar objects to the query object, where the similarity measure considers both spatial proximity and textual similarity. Reverse Spatial-Textual kNN (RSTkNN) queries have also been studied previously [13, 14] . An RSTkNN finds objects that have the query objects q as one of the k most spatial-textual relevant objects. An example application of reverse queries is to identify potential customers who consider a product as highly relevant for a service provider. This allows a service provider to satisfy customer preferences, and find locations close to potential customers. Now consider the following two example applications for spatial-textual queries:



(a) (b)

ID	Keywords
u_1	sushi, seafood
u_2	sushi
u_3	sushi, noodles
u_4	noodles
o_1	sushi
o_2	noodles

Figure 1: Example of a MaxBRSTkNN query.

EXAMPLE 1. *In social media advertising, a user is shown a limited number of advertisements that are highly relevant to her location and preferences (i.e., top-k relevant advertisements). In this case, the application is to find the location and the keywords to include in an advertisement such that it is displayed to the maximum number of users. For space and cost reasons, we assume that there is a limitation on the number of keywords per advertisement.*

EXAMPLE 2. *Consider an application that finds the optimal location to open a restaurant, and the items to include in the menu such that it will be a top-k restaurant for the maximum number of customers. In Figure 1a, the users $u_1, \ldots, u_4$ are shown with circles; o_1 and o_2 are existing restaurants. The table shows the corresponding text descriptions. Here, the top-*1 *spatial-textual relevant restaurant for each user is shown with a connecting dotted line. Suppose a service provider wants to open a new restaurant, o_x in one of the locations l_1, l_2, l_3 shown with triangles in Figure 1b, and for cost reasons, the number of menu items that can be displayed is '*1*'. The choices of the menu items are {*'`sushi`', '`seafood`', and '`noodles`'}. If o_x is placed in l_1, and the menu is '`sushi`', o_x becomes a top-*1 *relevant restaurant for u_1, u_2, and u_3, as shown in Figure 1b. For the given choices of locations and keywords, and the other competitor restaurants, o_x can be the top-*1 *relevant restaurant of a maximum of '*3*' users here. So in this example, the optimal location and menu item are l_1 and '`sushi`', respectively.*

In these examples, the underlying problem is to find the location and text description for a specific product or service such that

Proceedings of the VLDB Endowment, Vol. 9, No. 6
Copyright 2016 VLDB Endowment 2150-8097/16/02.

the product is one of the top-k most relevant objects of the maximum number of users, i.e., maximizing the size of the bichromatic RSTkNN (BRSTkNN) of the product. Here, the top-k ranking is based on both spatial proximity and textual relevance. In this paper, we study this new query and refer to it as a *Maximized Bichromatic Reverse Spatial Textual k Nearest Neighbor (*MaxBRSTkNN*)* query. To the best of our knowledge, we are the first to explore this query in the spatial-textual database domain.

DEFINITION 1. *Let D be a bichromatic dataset, where U is a set of users and O is a set of objects. Each $u \in U$ and each $o \in O$ is associated with a location and a set of keywords. A* MaxBRSTkNN *query $q(o_x, L, W, w_s, k)$ over D, where o_x is a spatial-textual object, L is a set of candidate locations, W is a set of candidate keywords, w_s is a positive integer where $w_s \leq |W|$, and the number of relevant objects to be considered k, finds a location $\ell \in L$ and a set of keywords $W' \subseteq W$, $|W'| \leq w_s$ such that if o_x is located in ℓ and W' is the text description of o_x, the cardinality of the results of* BRSTkNN *of o_x is maximized. If o_x has an existing text description then $W' \cup$ (existing text of o_x) and ℓ combine to maximize* $|$BRSTkNN$|$ *of o_x. Otherwise, W' becomes the text description of o_x.*

Existing work for the MaxBRkNN problem focuses on spatial proximity only, and rely on different spatial properties such as intersecting geometric shapes [21, 20], space partitioning [26], or sweep-line techniques [12, 11] (more details in Section 2.1). Therefore, these approaches cannot be directly applied to the textual similarity part of our problem. Moreover, the existing index and solutions for answering RSTkNN queries [13, 14] are for the monochromatic case only – both the data objects and the query objects belong to the same type. In contrast, the MaxBRSTkNN query is bichromatic since there are two types of objects in the dataset, namely the users U and the objects O. Therefore, the techniques to answer the RSTkNN queries are not applicable to the MaxBRSTkNN problem. See Section 2.2 for further details.

Contributions. The key contributions of this research are:

1. We introduce a new type of query, MaxBRSTkNN, which finds a location and a set of keywords for an object so that the BRSTkNN of the object is maximized.
2. We prove that the MaxBRSTkNN query is NP-hard.
3. We develop approximate and exact algorithms to answer the query using effective dynamic pruning strategies, based on new spatial-textual indexes.
4. We present an efficient method to compute the top-k objects jointly, which is essential to improve the overall performance and of independent interest.
5. We conduct an extensive experimental study to show the efficiency of our approach using two datasets.

2. RELATED WORK

In this section we review previous work in the areas spatial databases, spatial-textual queries, and top-k query processing.

2.1 Spatial databases

Relevant work from the spatial database domain can be categorized mainly as *Maximizing Bichromatic Reverse k Nearest Neighbor* (MaxBRkNN) queries and location selection queries.

MaxBRkNN queries. Wong et al. [21] introduced the MAXOVERLAP algorithm to solve the MaxBRkNN problem. The algorithm iteratively finds the intersection point of the Nearest Location Circles (NLCs) that are covered by the largest number of NLCs. The optimal region is the overlap of these NLCs. This work also supports ℓ-MaxBRkNN queries to find the ℓ best regions. In a later work, Wong et al. [20] extended the MAXOVERLAP algorithm to support the L_p-norm and three-dimensional space. However, the scalability of MAXOVERLAP is an issue, as the computation of the intersection points for the NLCs is expensive.

Other works exist that overcome the limitations of MAXOVERLAP. Zhou et al. [26] introduced the MAXFIRST algorithm which iteratively partitions the space into quadrants and use the NLCs to prune the quadrants that cannot be a part of the result. Liu et al. [12] present MAXSEGMENT algorithm that transforms the optimal region search problem to the optimal interval search problem. They use a variant of plane sweep to find the optimal interval.

Approximate solutions have also been proposed to improve the efficiency. Lin et al. [11] propose OPTREGION where each NLC is approximated by the minimum bounding rectangle (MBR) and a sweep-line technique is used to find the overlapping NLCs. An estimation of the number of overlapping NLCs is computed using the MBRs to prune the intersection points. Alternately, Yan et al. [22] propose a grid-based approximation algorithm called FILM. Since the algorithm is approximate, the solution requires an order of magnitude less computation time than MAXOVERLAP. The authors extended FILM to answer the related problem of locating k new services that collectively maximize the total number of users.

These works focus solely on spatial properties such as the intersection of geometric shapes [21, 20], space partitioning [26], or sweep-line techniques [12, 11] in the query processing methods. Therefore, it is not straightforward to extend these solutions to support the textual component of the MaxBRSTkNN query.

Location selection queries. The works [8, 24] present the optimal location queries, which find a location for a new facility that minimizes the average distance from each customer to the closest facility. Zhang et al. [24] propose the *MDOL prog* algorithm that partitions the space to find the optimal location. The work [8] maintain the *influence set* of a potential location p that includes the customers for whom the nearest facility distance is reduced if a new facility is established at p. A similar problem is presented in [7] to find a location for a new server such that the maximum distance between the server and any client is minimized. Huang et al. [6] explore the *maximal influence query*, where the influence of a location p represents the cardinality of customers whose corresponding nearest facility will be p if a new facility is established in p. A maximal influence query finds the optimal location to place a new facility such that the influence of that facility is maximized.

These queries focus on an aggregation over distances from the query location, e.g., the average or the minimum distance. These works do not directly address maximizing the BRkNNs problem.

2.2 Spatial-textual databases

In the literature of spatial-textual queries, the reverse spatial-textual kNN (RSTkNN) [13, 14] is the most relevant to our problem.

RSTkNN. Given a dataset D of spatial-textual objects, a target query object q, the RSTkNN query finds all the objects in D that have q in their list of top-k relevant objects. The ranking of the objects use an objective function which combines both spatial proximity and text relevancy. Lu et al. [13] propose the Intersection-Union R-tree (IUR-tree) index, and later present a cost analysis in [14] for RSTkNN queries. Each node of an IUR-tree consists of an MBR and two textual vectors: an intersection vector and a union vector. The weight of each term in the intersection (union) textual vector is the minimum (maximum) weight of the terms in the documents that are contained in the corresponding subtree. Each non-leaf node is also associated with the number of objects in the subtree rooted at that node.

In their proposed solution, an upper and a lower bound estimation of similarity is computed between each node of the IUR-tree and the k·th most similar object. A branch-and-bound algorithm is then used to answer the RSTkNN query. In this work, the computation of the bounds and the algorithm are designed for the monochromatic case only since both the data objects and the query objects belong to the same type, and the nodes of the tree store only one type of object.

2.3 Other top-k maximizing queries

Chen-Yi et al. [2] select k products from a set of candidates such that the expected number of customers is maximized. Each customer is associated with a requirement vector, and a customer chooses any product that satisfies the requirement with equal probability. This work does not consider the ranking of the product. Vlachou et al. [18] determine the k most influential objects, where influence of an object is the cardinality of the reverse top-k. The ranking is based on a query weight vector. Koh et al. [9] select at most k products from a set of candidates such that the total number of customers considering them as their top-t relevant products is maximized. They assume a product is composed of n different types of query independent components. Wan et al. [19] explore a similar problem where given a set of existing products, a set of new products is generated such that the products are not dominated by the existing ones. In contrast, the candidates of the MaxBRSTkNN problem are query parameters that are processed in query time.

3. PRELIMINARIES

Let D be a bichromatic dataset, where U is a set of users and O is a set of objects. Each object $o \in D$ is a pair $(o.l, o.d)$, where $o.l$ is the spatial location (point, rectangle, etc.) and $o.d$ is a set of keywords. Each user $u \in U$ is also defined as a similar pair $(u.l, u.d)$. Recall from Definition 1 that a MaxBRSTkNN query returns a location $\ell \in L$ and a set $W' \subseteq W$, $|W'| \leq w_s$ such that, if $o_x.l = \ell$ and $o_x.d = W' \cup o_x.d$, the size of BRSTkNN of o_x from U is maximized. An object o is ranked based on a combined score of spatial proximity and textual relevance with respect to a user u, specifically, using the following equation:

$$STS(o,u) = \alpha \cdot SS(o.l,u.l) + (1-\alpha) \cdot TS(o.d,u.d), \quad (1)$$

where $SS(o.l,u.l)$ is the spatial proximity between locations, the textual relevance is $TS(o.d,u.d)$, and the preference parameter $\alpha \in [0,1]$ defines the importance of one relevance measure relative to the other. The value of both measures are normalized within $[0,1]$. Here, a higher score denotes higher relevance.

Spatial proximity. The spatial proximity of an object o with respect to a user u is: $SS(o.l,u.l) = 1 - \frac{dist(o.l,u.l)}{d_{max}}$, (2) where $dist(o.l,u.l)$ is the minimum Euclidean distance, and d_{max} is the maximum Euclidean distance between any two points in D.

Text relevance. In this problem, an object o is considered relevant to a user u iff $o.d$ contains at least one term $t \in u.d$. Several measures can be used to compute the relevance between any two text descriptions [15]. We present two commonly used text relevance metrics and a non-metric measure in the following.

The **TF-IDF metric** weighs a term in a document based on term frequency (TF) and inverse document frequency (IDF). The TF, $tf(t,d)$, is the number of times term t appears in document d, and $idf(t,O) = \log \frac{|O|}{|d \in O, tf(t,d)>0|}$ measures the importance of t in the set O. Here, the text relevance of an object o with respect to a user u is $TS(o.d,u.d) = \sum_{t \in u.d} tf(t,o.d) \times idf(t,O)$.

In **Language Model (LM)**, the weight of a term t in the text description of an object, $o.d$ is defined as:

$$\hat{p}(t|\theta_{o.d}) = (1-\lambda)\frac{tf(t,o.d)}{|o.d|} + \lambda \frac{tf(t,C)}{|C|}, \quad (3)$$

where C is the concatenation of all documents in the collection, $\frac{tf(t,C)}{|C|}$ is the maximum likelihood estimate of t in C, and λ is a smoothing parameter for Jelinek-Mercer smoothing. [23] discuss the appropriate settings of λ, which is used to adjust the maximum likelihood estimator due to the data sparseness. In LM, the text relevance of an object o with respect to a user u is:

$$TS(o.d,u.d) = \frac{\sum_{t \in u.d} \hat{p}(t|\theta_{o.d})}{P_{max}}, \quad (4)$$

where P_{max} is used to normalize the score into the range from 0 to 1 and is computed as, $P_{max} = \sum_{t \in u.d} \max_{o' \in D} \hat{p}(t|\theta_{o'.d})$.

A non-metric text similarity measure is the **Keyword Overlap (KO)**, which is the intersection of the user and object keywords. Formally, the Keyword Overlap is $TS(o.d,u.d) = \frac{|u.d \cap o.d|}{|u.d|}$, where $|u.d|$ is used to normalized the score in $[0,1]$.

In this paper, we describe our algorithm using the Language Model. However, our approaches are applicable for any text-based relevance measure. In Section 8, we compare the performances of our approaches for different text similarity measures.

NP-Hardness. We now show that the MaxBRSTkNN problem is NP-hard by reduction from the Maximum Coverage problem.

LEMMA 1. *The* MaxBRSTkNN *problem is NP-hard.*

PROOF. Given a collection of sets $S = \{S_1, S_2, \ldots, S_m\}$ over a set of objects O, where $S_i \subseteq O$, and a positive integer p, the Maximum Coverage (MC) problem is to find a subset $S' \subseteq S$ such that $|S'| \leq p$ and the number of covered elements by S', i.e., $|\cup_{S_i \in S'} S_i|$ is maximized. The MC problem is NP-hard [5].

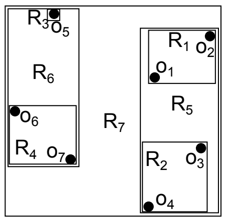
Consider a special case of the MaxBRSTkNN problem where $\alpha = 1$. Here, the similarity score of the objects that contain at least one of the user keywords are measured by the spatial proximity using Equation 1. Also assume that the number of candidate locations, $|L| = 1$ in this special case. So the location of $o_x.l = \ell$, where ℓ is the only candidate location in L.

For each candidate keyword $w_i \in W$, let BRSTkNN (ℓ, w_i) be the set of users that have o_x as a top-k object when w_i is included in $o_x.d$. So we get a collection of the set of BRSTkNN users, one for each $w_i \in W$. The goal of the MaxBRSTkNN query is to select at most w_s number of keywords from W, such that the total number of BRSTkNN users is maximum. That is, solving the maximum coverage problem is equal to finding a subset of the candidate keywords, $W' \subseteq W$, where $|W'| \leq w_s$ maximizes $|\cup_{w_i \in W'}$ BRSTkNN$(\ell, w_i)|$.

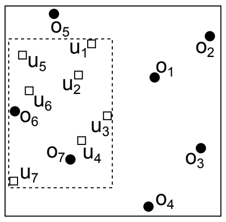
Again, BRSTkNN (ℓ, w_i) for each $w_i \in W$ corresponds to each set S_i of the MC problem, where each user $u_j \in U$ corresponds to the object $o_j \in O$. Therefore, finding a subset W' of the candidate keywords where $|W'| \leq w_s$ maximizing $|\cup_{w_i \in W'}$ BRSTkNN$(\ell, w_i)|$ is equivalent to solving the maximum coverage problem. □

4. BASELINE APPROACH

As MaxBRSTkNN is a new type of query, a first attempt at a solution is to issue a BRSTkNN query for all possible tuples $\langle \ell, c \rangle$, where ℓ is a candidate location from L, and c is a combination of w_s number of keywords from W. Then we return the tuple with the maximum BRSTkNNs. However, prior work on RSTkNN addressed only the monochromatic case. So, there is no obvious way to extend these approaches to MaxBRSTkNN queries. Another possible approach is to solve the spatial component using one of the solution



(a) Objects and MBRs



(b) Objects and users

Figure 2: The placement of the objects and the users

for the MaxBRkNN problem, and solve the textual component separately. Then, the results can be combined to get the final answer. But from the proof of Lemma 1, we know that the textual part of the MaxBRSTkNN problem alone is NP-hard, even when there is only one candidate location. So this extension cannot provide an efficient solution for the MaxBRSTkNN problem.

Based on lessons learned in prior related work, we can however propose a reasonable baseline for MaxBRSTkNN queries. In this baseline, the important components are as follows.

Computing top-k objects. We assume all objects $o \in O$ are stored on disk and indexed using a spatial-textual index. We use the IR-tree [3] to index O. The top-k objects for each user $u \in U$ are computed using an IR-tree as originally described in [3]. Thus, the relevance score $RS_k(u)$ of the k·th ranked object of each user u is obtained.

Selecting the best candidate. The set of keyword combinations C of w_s keywords from W are generated. For each candidate location $\ell \in L$, and each keyword combination $c \in C$, the total relevance score of ℓ and $o_x.d \cup c$ are computed for the users $u \in U$, where $(o_x.d \cup c) \cap u.d \neq \emptyset$. If this score is greater than $RS_k(u)$, u is a BRSTkNN for the tuple $\langle \ell, c \rangle$. In each iteration, the tuple that has the maximum number of BRSTkNNs so far is tracked. Finally, the best tuple is returned as the result. Note that this method returns exactly w_s number of keywords in the result.

Limitations. The above method is computationally expensive for several reasons: (i) computing the top-k results for *all* the users; (ii) iterating the process for *all* the candidate locations; (iii) generating all the combinations, C of w_s number of candidate keywords; and (iv) computing the relevance scores of all the tuples of the candidate locations, and $c \in C$ for each user $u \in U$. In addition, we need to read all the users into memory to compute the top-k objects.

Moreover, the number of combinations of size w_s from the candidate keywords is $N_k = \binom{|W|}{w_s}$, and the total number of tuples of the candidate locations and keywords is $|L| \times N_k$. As this number is combinatorially large, scanning all combinations is not practical for a large number of candidates. To overcome the limitations in the baseline approach, we seek techniques that:

- Efficiently compute the top-k objects for the users;
- Avoids computing the top-k objects for the users that cannot affect the result; and
- Prune the candidate locations and keywords that cannot be part of the result.

5. JOINT TOP-K PROCESSING

To answer a MaxBRSTkNN query in the baseline approach, the score of the k·th ranked object for each user must be determined. Computing the top-k objects for each user requires retrieving the objects from disk, which is not I/O efficient. Moreover, the same objects might be retrieved multiple times for different users. To overcome this drawback, we compute the top-k objects of the users jointly using different pruning strategies, and ensure that an object is retrieved only once. Thus, we can reduce the overall computational costs by sharing processing and I/Os. We first present the indexing method that we use to index the disk resident object set O, and then we describe the algorithm. The frequently used symbols are summarized in Table 1.

Table 1: Notation

Symbol	Description
L	The set of candidate locations
W	The set of candidate keywords
o_x	The input object, for which a location and a set of keywords are selected from candidates
W'	The subset of W to return
w_s	The at most number of candidate keywords to return
$RS_k(u)$	Relevance score of the k·th ranked object of user u
u_s	Super-user
LU_ℓ	List of users that can be BRSTkNN for location ℓ
$max\ell$	The location with the maximum LU_ℓ

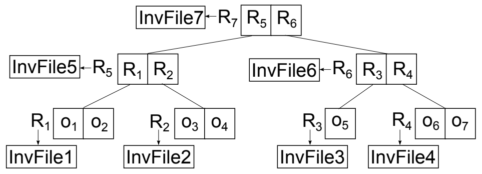


Figure 3: Min-max IR-tree (MIR tree)

5.1 Index Structure

We first give an brief overview of the IR-tree [3]. Next, we describe how we have extended the IR-tree, denoted as the MIR-tree, to index the set of objects O for the joint top-k processing.

IR-tree. The IR-tree is an R-tree where each node has a reference to an inverted file. Each node R contains entries of the form $(cp, rect, cp.di)$. If R is a leaf node, cp refers to an object $o \in O$, $rect$ is the bounding rectangle of o, and $cp.di$ is an identifier of the text description. Otherwise, cp refers to a child node of R, $rect$ is the MBR of all entries of that child node, and $cp.di$ is the identifier of a pseudo-document. The pseudo-document is the union of all text descriptions in the entries of the child node, where the weight of a term t is the maximum weight of the documents contained in the subtree. Each node has a reference to an inverted file for the entries stored in the node. A posting list of a term t in the inverted file is a sequence of pairs $\langle d, w_{d,t} \rangle$, where d is the document id containing t, and $w_{d,t}$ is the weight of t in d.

Min-max IR-tree (MIR-tree). We propose the Min-max IR-tree (MIR-tree) to index the objects. The objects are inserted in the same manner as in IR-tree. However, unlike an IR-tree, each term is associated with both the maximum $maxw_{d,t}$ and minimum $minw_{d,t}$ weights in each document. The posting list of a term t is a sequence of tuples $\langle d, maxw_{d,t}, minw_{d,t} \rangle$, where d is the document id containing t, $maxw_{d,t}$ is the maximum, and $minw_{d,t}$ is the minimum weight of term t in d, respectively. If R is a leaf node, both weights are the same as the weight of term t, $w_{d,t}$ in the IR-tree. If R is a non-leaf node, the pseudo-document of R is the union of all text descriptions in the entries of the child node. The maximum (minimum) weight of a term t in the pseudo-document is the maximum (minimum) weight in the union (intersection) of the documents contained in the subtree. If a term is not in the intersection, $minw_{d,t}$ is set to 0.

Figure 2a shows the objects $O = \{o_1, o_2, \ldots, o_7\}$ where the MBRs are constructed according to the IR-tree. Figure 3 illustrates the MIR-tree for O. Table 2 presents the inverted files of the leaf nodes (InvFile 1, InvFile 2, InvFile 3, and InvFile 4) and the non-leaf nodes (InvFile 5, InvFile 6, and InvFile 7). As a specific example,

Table 2: Posting lists of the example MIR-tree

Term	InvFile 1	InvFile 2	InvFile 3	InvFile 4	InvFile 5	InvFile 6	InvFile 7
t_1	$(o_1,1,1)$	$(o_3,5,5)$	$(o_5,4,4)$	$(o_6,1,1),(o_7,2,2)$	$(R_1,1,0)$, $(R_2,5,0)$	$(R_3,4,4)$, $(R_4,2,1)$	$(R_5,5,0),(R_6,4,1)$
t_2	$(o_1,4,4)$	-	$(o_5,1,1)$	-	$(R_1,4,0)$	$(R_3,1,1)$	$(R_5,4,0),(R_6,1,0)$
t_3	-	$(o_3,5,5)$	-	$(o_6,1,1)$	$(R_2,5,0)$	$(R_4,1,0)$	$(R_5,5,0),(R_6,1,0)$
t_4	$(o_2,1,1)$	$(o_4,2,2)$	-	$(o_7,3,3)$	$(R_1,1,0),(R_2,2,0)$	$(R_4,3,0)$	$(R_5,2,0),(R_6,3,0)$

the maximum (minimum) weight of term t_1 in entry R_4 of InvFile 6 is 2 (1), which is the maximum (minimum) weight of the term in the union (intersection) of documents (o_6,o_7) of the node R_4.

Cost analysis. In contrast to the original IR-tree [3], the space requirement of the MIR-tree includes an additional weight stored for the minimum text relevance for each term in each node. Specifically, for a node N, if the number of terms is M, the additional required space is $\sum_{i=1}^{M} d_i$, where d_i is the number of objects in the posting list of term t_i in node N. The construction time of the MIR-tree is very similar to the original IR-tree. During tree construction, when determining the maximum weight of each term in a node, the minimum weight of that term can be determined concurrently. As the splitting and merging of the nodes are executed in the same manner as the IR-tree, the update costs of the MIR-tree are also same as the IR-tree.

In this paper, the proposed MIR-tree is an extension of the original IR-tree presented in [3]. Cong et al. [3] also proposed other variants of the IR-tree, such as the DIR-tree and the CIR-tree, where both spatial and textual criteria are considered to construct the nodes of the tree. The same structures can be used to construct our proposed extension. For example, the nodes of the MIR-tree can be constructed in the same manner as the DIR-tree, and the posting lists of each node will contain both the minimum and maximum weights of the terms. Due to the page limitations, we do not include the details of the other variants of the IR-tree here.

5.2 Grouping of Users

Our goal is to access the necessary objects from disk, and avoid duplicate retrieval of objects for different users. We form a group of the users for this purpose, denoted as a "super-user" (u_s), and the objects are accessed using this group instead of individual users.

Construction of super-user. The "super-user" (u_s) is constructed such that $u_s.l$ is the MBR enclosing the locations of all users, $u_s.dUni$ is the union, and $u_s.dInt$ is the intersection of the keywords of all users, respectively. As an example, Figure 2b shows the locations of the users $U = u_1, u_2, \ldots, u_7$ and the corresponding text descriptions are presented in Table 3. The location of the "super-user", $u_s.l$ is the MBR enclosing the locations of all the users, shown with a dotted rectangle. Here, the intersection of the keywords of all the users, $u_s.dInt$ is '1000' and the union, $u_s.dUni$ is '1111'.

Table 3: Text description of the users

Term \ User	u_1	u_2	u_3	u_4	u_5	u_6	u_7
t_1	1	1	1	1	1	1	1
t_2	0	0	0	1	1	0	0
t_3	1	1	0	1	0	0	0
t_4	1	1	0	0	0	1	1

We now present the notion for upper and lower bound estimations for spatial-textual relevance scores between any user u, and any object node of the MIR-tree using this super-user.

5.3 Upper and Lower Bound Estimation

The maximum spatial-textual similarity between any node E of the MIR-tree and the super-user u_s is computed as:

$$MaxSTS(E,u_s) = \alpha \cdot MinSS(E.l,u_s.l) + (1-\alpha) \cdot MaxTS(E.d,u_s.dUni),$$

Algorithm 1: JOINT_ TOPK(MIR-tree, U, k)

```
1.1  Output: The top-k objects of all users
1.2  $u_s.l \leftarrow_{\forall u \in U} MBR(u.l)$; $u_s.dUni \leftarrow_{\forall u \in U} \cup(u.d)$; $u_s.dInt \leftarrow_{\forall u \in U} \cap(u.d)$
1.3  Initialize max-priority queue $PQ, RO$, min-priority queue $LO$
1.4  $E \leftarrow$ MIR-tree($root$)
1.5  ENQUEUE($PQ, E, LB(E,u_s)$)
1.6  while $PQ \neq \varnothing$ do
1.7      $E \leftarrow$ DEQUEUE($PQ$)
1.8      if $E$ is leaf then
1.9          if $|LO| < k$ then
1.10             ENQUEUE($LO, E, LB(E,u_s)$)
1.11             if $|LO| = k$ then
1.12                 $RS_k(u_s) \leftarrow LB(\text{TOP}(LO), u_s)$
1.13         else if $UB(E,u_s) \geq RS_k(u_s)$ then
1.14             ENQUEUE($LO, E, LB(E,u_s)$)
1.15             $Obj \leftarrow$ DEQUEUE($LO$)
1.16             $RS_k(u_s) \leftarrow LB(Obj, u_s)$
1.17             if $UB(Obj,u_s) \geq RS_k(u_s)$ then
1.18                 ENQUEUE($RO, Obj, UB(Obj,u_s)$)
1.19     else
1.20         if $|LO| < k$ or $UB(E,u_s) \geq RS_k(u_s)$ then
1.21             for each element $e$ of $E$ do
1.22                 ENQUEUE($PQ, e, LB(e,u_s)$)
1.23 return INDIVIDUAL_TOPK($U, u_s, LO, RO$)
```

where $MinSS$ is computed from the minimum Euclidean distance between the two MBRs using Equation 2, and $MaxTS$ is the maximum textual similarity between $E.d$ and the union of the keywords of the users, $u_s.dUni$ computed as:

$$MaxTS(E.d,u_s.dUni) = \frac{\sum_{t \in u_s.dUni}(maxw_{E.d,t})}{P_{max}}.$$

where $maxw_{E.d,t}$ is the maximum weight of the term t in the associated document of node E. As described in Section 5.1, if E is a non-leaf node, $maxw_{E.d,t}$ is the maximum weight in the union of the documents contained in the subtree of E. Otherwise, $maxw_{E.d,t}$ is the weight of term t in document $E.d$ computed using Equation 3 in language model.

We now present a lemma that enables us to estimate an upper bound on the relevance between any user $u \in U$, and any object node E using the super-user u_s, where E is a node of the MIR-tree.

LEMMA 2. *$\forall u \in U$, $MaxSTS(E,u_s)$ is an upper bound estimation of $STS(E,u)$. For any object node E, $STS(E,u) \leq MaxSTS(E,u_s)$.*

PROOF. Recall that the $u_s.l$ of the super user is the MBR of the locations for all of the users in U. For an object node E of the MIR-tree, the $MinSS(E,u_s)$ is the minimum Euclidean distance between the two MBRs of E and u_s using Equation 2. As the location $u.l$ of any user $u \in U$ is inside the rectangle $u_s.l$, the value $SS(E,u)$ must be less than or equal to $MinSS(E,u_s)$. For textual similarity, as $u_s.dUni = \cup_{u \in U} u.d$, the maximum textual similarity score between any user $u \in U$, and any object node E that can be achieved is $MaxTS(E,u_s)$ from Equation 4. Since the spatial-textual score $STS(E,u)$ is the weighted sum of the corresponding spatial and textual scores, $\forall u \in U$, the score $STS(E,u)$ must also be less than or equal to $MaxSTS(E,u_s)$. □

Lemma 2 presents that MaxSTS(E,u_s) is a correct upper bound estimation of relevance between a node E of the MIR-tree and any

$u \in U$, as the relevance $\mathrm{STS}(E,u)$ is always less than $\mathrm{MaxSTS}(E,u_s)$. Similarly, a lower bound relevance can be computed as: $LB(E,u_s) = \alpha \cdot MaxSS(E.l, u_s.l) + (1-\alpha) \cdot MinTS(E.d, u_s.dInt)$, where *MaxSS* is computed from the maximum Euclidean distance between the two MBRs, *MinTS* is the minimum textual relevance between E and u_s*.dInt* computed using the minimum weights of the terms in E. Similar to the upper bound estimation, we can prove that the property $\forall u \in U, STS(E,u) \geq LB(E,u_s)$ always holds.

5.4 Algorithm

We assume the set of objects O resides on disk and is indexed using an MIR-tree. The main idea joint top-k processing is to reduce the number of I/O operations by sharing the I/Os among users and accessing the necessary objects and tree nodes only once. This is achieved by: a) a careful tree traversal; and b) an efficient top-k object computation of the individual users. We utilize the super-user to access the MIR-tree and share I/Os, and the bounds of relevance to prune the nodes that do not contain any top-k object of any user. The pseudocode of the tree traversal step in the joint top-k processing is shown in Algorithm 1. Finally, the top-k results of the individual users are computed by applying several pruning strategies as presented in Algorithm 2.

Tree traversal. The pseudocode is presented in Algorithm 1. Here, the MIR-tree is traversed for the super-user u_s instead of the individual users. Line 1.2 shows the construction of u_s from U. Initially, a max-priority queue PQ is created (Line 1.3) to keep track of the nodes that are yet to be visited, where the key is the lower bound w.r.t. u_s. We also maintain a min-priority queue LO (Line 1.3) to keep k number of objects with the best lower bounds found so far. Lines 1.9-1.12 show how LO is initially filled up with k objects according to their lower bounds. We use actual objects instead of object nodes in LO for better estimation of relevance. We also store the k·th best lower bound relevance score, $RS_k(u_s)$ found so far.

Since the score $RS_k(u_s)$ is the k·th best lower bound score for any user $u \in U$, and any unseen object o, the similarity score must be greater than or equal to $RS_k(u_s)$ for o to be one of the top-k objects of u. Therefore, we need to consider only those nodes E, where $UB(E,u_s) \geq RS_k(u_s)$. If E is an object satisfying this condition, then LO is adjusted such that it contains k objects with the best lower bounds. The score $RS_k(u_s)$ is also updated accordingly. If the object *Obj* dequeued from LO in this adjustment process, has better upper bound than the updated $RS_k(u_s)$, *Obj* is stored in a priority queue RO (shown is Lines 1.13-1.18). Here, RO is a max-priority queue where the key is the upper bound similarity score w.r.t. u_s. If a non-leaf node E cannot be pruned, the entries of E are retrieved from disk and enqueued in PQ as shown in Lines 1.20-1.22. Finally, the objects in LO and RO are used to compute the top-k objects of individual users in a later step (Line 1.23).

We traverse the MIR-tree according to the lower bound in descending order so that the objects with the best lower bounds will be retrieved early, thereby enabling better pruning. Next, we present an example to explain the procedure of the tree traversal step.

EXAMPLE 3. *The object $O = o_1, o_2, \ldots, o_7$ in Figure 2a are indexed with an* MIR-tree *as shown in Figure 3. The users $U = u_1, u_2, \ldots, u_7$ are shown in Figure 2b, where the dotted box is the MBR of the users, i.e., $u_s.l$. Table 2 and Table 3 present the text descriptions. Let $k = 1$, the tree traversal step starts by enqueueing the root node R_7 in PQ, and then performing the following steps:*

1. *Dequeue R_7, $PQ : (R_6, 0.6), (R_5, 0.3)$*
2. *Dequeue R_6, $PQ : (R_4, 0.6), (R_3, 0.5), (R_5, 0.3)$*
3. *Dequeue R_4, $PQ : (o_7, 0.7), (R_3, 0.5), (o_6, 0.5), (R_5, 0.3)$*
4. *Dequeue o_7, as $|LO| < k$, $LO : o_7$, $RS_k(u_s) = 0.7$*
5. *Dequeue R_3, as $UB(R_3, u_s) = 0.8$, enqueue o_5 in PQ $PQ : (o_5, 0.7), (o_6, 0.5), (R_5, 0.3)$*
6. *Dequeue o_5, as $UB(o_5, u_s) = 0.8$, enqueue o_5 in RO $LO : o_7$, $RO : o_5$, $RS_k(u_s) = 0.7$; $PQ : (o_6, 0.5), (R_5, 0.3)$*
7. *Dequeue o_6, as $UB(o_6, u_s) = 0.9$, enqueue o_5 in RO $L_O : o_7$, $R_O : o_6, o_5$, $RS_k(u_s) = 0.7$; $PQ : (R_5, 0.3)$*
8. *Dequeue R_5, as $UB(R_5, u_s) = 0.6 < RS_k(u_s)$, discard.*

Top-k object of individual users. In the tree traversal step, the priority queues LO and RO store all the objects that can be a top-k object of at least one user in U. Therefore, considering only the objects in LO and RO is sufficient to obtain the top-k objects for all $u \in U$. Algorithm 2 summarizes this process. For each $u \in U$, a min-priority queue H_u of objects is initialized (Line 2.1) where the key is the total relevance score of the object w.r.t. the user u. For each u, the relevance score between each element $o \in LO$ and u is computed and inserted in H_u. The score of the k·th ranked object of a user u, $RS_k(u)$ computed so far is also stored (Line 2.2-2.5).

The objects of RO can be pruned in two steps. First, if the upper bound score of an object $o \in RO$ w.r.t. u_s is less than $RS_k(u)$, then o cannot be a top-k object of u. The subsequent objects of such o in RO can also be pruned from consideration (Lines 2.7) as RO is maintained in ascending order of the upper bound scores w.r.t. u_s. Otherwise, if $STS(o,u) \geq RS_k(u)$, o is inserted in H_u. H_u is adjusted such that it contains k number of objects with the best relevance scores, and $RS_k(u)$ is updated accordingly as shown in Lines 2.9-2.11. Finally, the priority queue H_u of each user $u \in U$ contains its top-k objects in reverse order and $RS_k(u)$ is the spatial-textual similarity score of the k·th ranked object of the corresponding user.

EXAMPLE 4. *Continuing the example of the previous step, consider u_6. Initially, $LO : (o_7, 0.7, 0.9)$, $RO : (o_6, 0.5, 0.9)$, $(o_5, 0.7, 0.8)$, where the entries are presented as (ID, LB, UB) w.r.t. u_s. First, object o_7 from LO is considered. As $\mathrm{STS}(o_7, u_6) = 0.75$, so, $\mathrm{RS}_k(u_6)$ becomes 0.75 and $H_{u_6} : o_7$. Then the objects in RO are considered. Here, $\mathrm{STS}(o_6, u_6) = 0.85$, so $\mathrm{RS}_k(u_6)$ becomes 0.85 and $H_{u_6} : o_6$. As the upper bound of the next object of R_O, $\mathrm{UB}(o_5, u_s) < \mathrm{RS}_k(u_6)$, we stop processing for u_6. The top-1 object of u_6 is o_6, where $\mathrm{RS}_k(u_6) = 0.85$. The process is repeated for all $u \in U$.*

Here, we have presented an efficient process to compute the top-k objects of all the users. Now the selection of the best tuple of candidate location and keywords is described.

Algorithm 2: INDIVIDUAL_TOPK(U, u_s, LO, RO)

```
2.1  Initialize an array $H$ of $|U|$ min-priority queues for each $u \in U$.
2.2  for each $u \in U$ do
2.3      for each $o \in LO$ do
2.4          ENQUEUE($H_u, o, STS(o,u)$)
2.5      $RS_k(u) \leftarrow STS(\mathrm{TOP}(H_u), u)$
2.6      for each $o \in RO$ do
2.7          if $UB(o,u_s) < RS_k(u_s)$ then break
2.8          else if $STS(o,u) \geq RS_k(u)$ then
2.9              ENQUEUE($H_u, o, STS(o,u)$)
2.10             DEQUEUE($H_u$)
2.11             $RS_k(u) \leftarrow STS(\mathrm{TOP}(H_u), u)$
2.12 return $H$
```

6. CANDIDATE SELECTION

As shown in Lemma 1, even when the number of candidate location is one, the candidate keyword selection process alone is NP-hard. Therefore, we propose a spatial-first pruning technique here.

6.1 Candidate location selection

Algorithm 3 shows the pseudocode of the steps to select the candidate location and keywords for the MaxBRSTkNN problem. Several pruning strategies are used in this process that utilize an upper and a lower bound estimation of relevance of the candidates. We present these bounds for the candidate locations in the following.

Upper bound estimation. For each $\ell \in L$, the upper bound relevance is computed in two steps, (i) w.r.t super-user u_s and (ii) w.r.t. each $u \in U$. The value $UBL(\ell, u_s)$ is computed such that for all $u \in U$, the relevance between o_x and u is at most $UBL(\ell, u_s)$, when $o_x.l = \ell$. The spatial upper bound is the minimum Euclidean distance between ℓ and u_s, as $u_s.l$ is the MBR for all of the users. For text relevance, a straightforward way is to consider the relevance as 1 (maximum), when the score is normalized within $[0,1]$. But we can achieve a tighter bound using the following lemma:

LEMMA 3. *Let W_h be the set of w_s number of keywords of the highest weights from $(u_s.dUni \cap W)$. The text relevance between o_x and a $u \in U$ after adding at most w_s number of candidate keywords is always less than or equal to the score $TS((o_x.d \cup W_h), u_s.dUni)$.*

PROOF. The text relevance between a user u and o_x can change by adding only the keywords that are present in $u.d$. As $u_s.dUni$ is the union of all $u.d$, the text relevance w.r.t any user u can be increased only by adding the candidate keywords that are present in $u_s.dUni$. Let w_1 and w_2 be two keywords in W_h where the weight of w_1 is greater than the weight of w_2 and $w_s = 1$. If a user u has both w_1 and w_2 in the text description, then from Equation 4, the text relevance of o_x w.r.t u found by adding w_1 must be equal or greater than that obtained by adding w_2. Even if a user u does not have all the keywords of W_h in $u.d$, the lemma still provides an upper bound estimation of text relevance that can be achieved by adding w_s number of candidate keywords. $\square$

So, the upper bound estimation of relevance of a candidate location w.r.t. the super-user u_s is

$$UBL(\ell, u_s) = \alpha \cdot MinSS(\ell, u_s.l) + (1-\alpha) \cdot TS((o_x.d \cup W_h), u_s.dUni)\,.$$

Similarly, an upper bound estimation of a candidate location ℓ w.r.t. any particular user u can be computed as $UBL(\ell, u) = \alpha \cdot MinSS(\ell, u.l) + (1-\alpha) \cdot TS(o_x.d \cup W_u, u.d)$, where W_u is the set of w_s number of keywords of the highest weights from $(u.d \cap W)$.

Lower bound estimation. The spatial lower bound is computed using the maximum Euclidean distance in a similar manner. For text relevance, the minimum score is computed from the original text description of o_x, i.e., $o_x.d$ and the intersection of all the user keywords, $u_s.dInt$. So, the lower bound estimation of $\ell \in L$ w.r.t. the super-user u_s is:

$$LBL(\ell, u_s) = \alpha \cdot MaxSS(\ell, u_s.l) + (1-\alpha) \cdot TS(o_x.d, u_s.dInt)\,.$$

Now we explain the steps and the pruning strategies employed in Algorithm 3 in the following.

- If $UBL(\ell, u_s) < RS_k(u_s)$, then no user can be a BRSTkNN for $o_x.l = \ell$, so ℓ is discarded (Lines 3.3-3.4). Otherwise, $UBL(\ell, u)$ is computed for each user. If $UBL(\ell, u) \geq RS_k(u)$, then u can be a BRSTkNN when $o_x.l = \ell$. A list of such users, LU_ℓ is maintained for each ℓ, (Lines 3.5-3.7).
- Here, we exploit a *best-first traversal* technique. A max-priority queue Q_L of candidate locations is maintained according to the cardinality of LU_ℓ. In each iteration the location, $max\ell$, with the maximum LU_ℓ is selected (Line 3.9).
- The location and keyword set combination $\langle \ell, W' \rangle_{best}$ with the maximum number of BRSTkNNs is tracked. As $LU_{max\ell}$ is constructed based on an upper bound, so when the cardinality of $LU_{max\ell}$ is less than $|\text{BRST}k\text{NN}(\langle \ell, W' \rangle_{best})|$, we cannot get a better tuple from the subsequent entries of Q_L. Thus, the computation can be *early terminated* (Lines 3.10).
- If $LBL(max\ell, u_s) \geq RS_k(u_s)$, then all the users in LU_ℓ are the BRSTkNN for $o_x.l = max\ell$, irrespective of the keyword selection. Thus we *avoid computing the candidate keywords* for such condition (Lines 3.12-3.13).
- Otherwise, the best candidate keyword set, W' is determined for $max\ell$. An approximate or an exact method presented in the following section is used to select W' (Line 3.15). $\langle \ell, W' \rangle_{best}$ is updated accordingly (Lines 3.16-3.17).

Algorithm 3: SELECT_CANDIDATE(U, L, W, k)

```
3.1  Initialize a max-priority queue $Q_L$.
3.2  $\langle \ell, W' \rangle_{best} \leftarrow \varnothing$.
3.3  for each $\ell \in L$ do
3.4      if $UBL(\ell, u_s) \leq RS_k(u_s)$ then
3.5          for each $u \in U$ do
3.6              if $UBL(\ell, u) \leq RS_k(u)$ then $LU_\ell \leftarrow u$
3.7      ENQUEUE($Q_L, \ell, |LU_\ell|$)
3.8  while $Q_L \neq \varnothing$ do
3.9      $max\ell \leftarrow$ DEQUEUE($Q_L$)
3.10     if $|LU_{max\ell}| < |\text{BRST}k\text{NN}(\langle \ell, W' \rangle_{best})|$ then break
3.11     else if $LBL(max\ell, u_s) \geq RS_k(u_s)$ then
3.12         if $|LU_{max\ell}| > |\text{BRST}k\text{NN}(\langle \ell, W' \rangle_{best})|$ then
3.13             $\langle \ell, W' \rangle_{best} \leftarrow \langle max\ell, \varnothing \rangle$
3.14     else
3.15         $W' \leftarrow$ Find best candidate keyword set for $max\ell$ using
             approximate or exact method.
3.16         if $|\text{BRST}k\text{NN}(max\ell, W')| > |\text{BRST}k\text{NN}(\langle \ell, W' \rangle_{best})|$ then
3.17             $\langle \ell, W' \rangle_{best} \leftarrow \langle max\ell, W' \rangle$
3.18 return $\langle \ell, W' \rangle_{best}$
```

6.2 Candidate keyword selection

Recall that the best candidate keyword set that provides the maximum number of BRSTkNN has to be determined for $o_x.l = max\ell$ (Line 3.17) in Algorithm 3. As this is an NP-hard problem, we first develop an approximation algorithm. We also present an exact method that uses several pruning strategies.

6.2.1 *Approximate algorithm*

The candidate keyword selection problem is shown to be NP-hard in Lemma 1 using a reduction from the Maximum Coverage (MC) problem. For the MC problem, a greedy algorithm exists which is a $(1 - 1/e) \simeq 0.632$ approximation algorithm. In the MC problem, the input is a collection of sets $S = \{S_1, S_2, \ldots, S_m\}$ and a number p. The greedy algorithm chooses a set in each step which contains the largest number of uncovered elements until exactly p sets are selected. This greedy algorithm is shown in [4] is the best-possible polynomial time approximation algorithm for the MC problem. Inspired by this algorithm, we propose an approximate algorithm to select the candidate keywords in each iteration of our algorithm when $o_x.l = max\ell$. However, some preprocessing must be done before applying the greedy algorithm.

Preprocessing. For each $w \in W$, a list of users, LUW_w is maintained based on an upper bound estimation, such that if w is included in the best keyword set W', these users can be the BRSTkNN of the tuple $\langle max\ell, W' \rangle$. As the $LU_{max\ell}$ is already computed based on such upper bound, we need to consider only the users in $LU_{max\ell}$.

Let, $HW_{w,u}$ is the set of w_s number of highest weighed keywords including w from $W \cap u.d$. A user u can be a BRSTkNN of the tuple $\langle max\ell, HW_{w,u} \rangle$ when $STS(o_x, u) \geq RS_k(u)$, where $o_x.l = max\ell$ and $o_x.d = o_x.d \cup HW_{w,u}$. So, $\forall u \in LU_{max\ell}$, $\forall w \in (W \cap u.d)$, we generate the list of users as, $LUW_w \leftarrow \{\delta \times u\}$, where, $\delta = 1$ if $STS(o_x, u) \geq RS_k(u)$ for $o_x.l = max\ell$ and $o_x.d = o_x.d \cup HW_{w,u}$. Otherwise, $\delta = 0$, i.e., u is not included in LUW_w.

Approximating the best candidate keyword set. Recall that in the MC problem, given a collection of sets $S = S_1, S_2, \ldots, S_m$ and a number p, the objective is to find a subset $S' \subseteq S$ such that $|S'| \leq p$ and the number of covered elements by S', i.e., $|\cup_{S_i \in S'} S_i|$ is maximized. In our case, the collection of the sets are the collection of LUW_w for each w and p is w_s. The greedy approach of MC is applied in our problem to find the best set of candidate keywords W' of size w_s such that $|\cup_{w \in W'} LUW_w|$ is maximized. This set W' is returned as the best candidate keyword set for the location $max\ell$. As the users are included in the LUW_w based on an upper bound estimation, so the number of actual BRSTkNNs for the combination of $max\ell$ and W' is computed and used for comparison in Line 3.18.

6.2.2 Exact algorithm

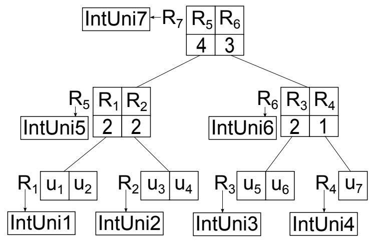
Algorithm 4: EXACT$(max\ell, LU_{max\ell}, W, w_s, k)$

```
4.1  $W_u \leftarrow_{\forall u \in LU_{max\ell}} \cup(u.d)$; $W' \leftarrow \varnothing$; $best \leftarrow 0$
4.2  if $|W \cap W_u| \leq w_s$ then $W' \leftarrow (W \cap W_u)$ else
4.3      $C \leftarrow$ combinations of $w_s$ number of keywords from $W \cap W_u$.
4.4      for each $c \in C$ do
4.5          for each $u \in$ LU$_{max\ell}$ do
4.6              if $LBL(max\ell, u) \geq RS_k(u)$ then
4.7                  BRST$k$NN$(max\ell, c) \leftarrow u$
4.8              else if $c \cap u.d \neq \emptyset$ then
4.9                  $Obj.l \leftarrow max\ell$
4.10                 $Obj.d \leftarrow (o_x.d \cup c)$
4.11                 if $STS(Obj, u) \geq RS_k(u)$ then
4.12                     BRST$k$NN$(max\ell, c) \leftarrow u$
4.13         if $|$BRST$k$NN$(max\ell, c)| > best$ then
4.14             $W' \leftarrow c$
4.15             $best \leftarrow |$BRST$k$NN$(max\ell, c)|$
4.16 return $W'$
```

The number of candidates can be small in some applications. Moreover, the search space can be pruned using several strategies when selecting the candidate keyword set. This motivates us to develop an exact algorithm for selecting the best keyword set W' of the MaxBRSTkNN query. The pseudocode is presented in Algorithm 4 and the pruning techniques are explained in the following.

- **Pruning users:** According to the definition of $UBL(\ell, u)$, only the users in $LU_{max\ell}$ can be the BRSTkNN of the tuple $\langle max\ell, W' \rangle$. So we need to consider only the users in $LU_{max\ell}$.
- **Pruning candidate keywords:** Let the union of the text description of the users in $LU_{max\ell}$ is W_u (Line 4.1). We need to consider only the candidate keywords that are contained in at least on of the users, i.e., $W \cap W_u$.
- **Early termination:** If $|W \cap W_u| \leq w_s$, this is the only possible candidate keyword set. So the process terminates and $W \cap W_u$ is returned as the best candidate keyword set for $max\ell$ as shown in Line 4.3-4.4.
- Here, the lower bound relevance, $LBL(\ell, u) = \alpha \cdot SS(\ell, u.l) + (1 - \alpha) \cdot TS(o_x.d, u.d)$, where $o_x.d$ is the original text description. If $LBL(max\ell, u) \geq RS_k(u)$, then u is a BRSTkNN for $max\ell$, regardless of the choice of keyword (Line 4.9-4.10).
- Let C is the set of combinations of w_s number of keywords from $W \cap W_u$. For a keyword combination $c \in C$, we process only those users where $c \cap u.d \neq \varnothing$.

7. INDEXING USERS



ID	Int.	Union
	$(t_1t_2t_3t_4)$	$(t_1t_2t_3t_4)$
IntUni1	1 0 1 1	1 0 1 1
IntUni2	1 0 0 0	1 1 1 0
IntUni3	1 0 0 0	1 1 0 1
IntUni4	1 0 0 1	1 0 0 1
IntUni5	1 0 0 0	1 1 1 1
IntUni6	1 0 0 0	1 1 0 1
IntUni7	1 0 0 0	1 1 1 1

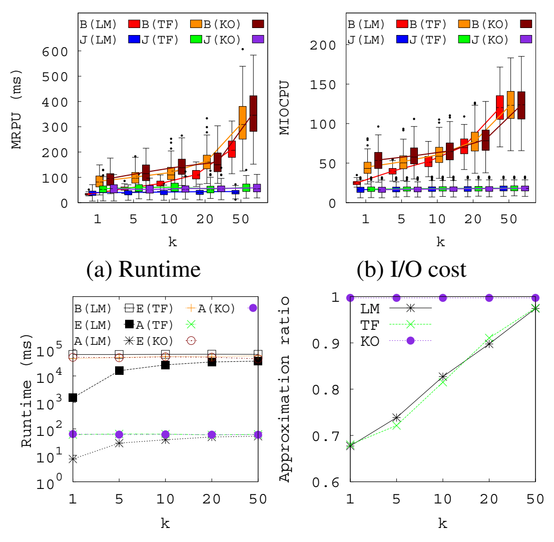
Figure 4: Example of Modified IUR-tree (MIUR tree)

In the previous sections we assume that the users reside in main-memory. If the number of users is large, we may want the user information to reside on disk. We also computed the top-k objects for every user in the previous configurations, which is computationally expensive. In this section we present an approach where the users are stored on disk and indexed using MIUR-tree, an extension of the IR-tree. The aim is to avoid computing the top-k objects of the users that do not have any affect on the result of MaxBRSTkNN query. The approach is also useful when the locations of the users are very sparse, therefore a hierarchy of users have a higher pruning capacity than a single super-user.

Modified IUR-tree (MIUR-tree). An MIUR-tree is essentially an R-tree where each node is augmented with the union and the intersection vector of the keywords appearing in the subtree. Each node is also associated with the number of actual objects stored in the subtree. Each leaf node R contains entries of the form $(cp, rect, cp.IntUnidi)$, where $cp.IntUnidi$ refers to the vector of the text description of o. If R is a non-leaf node then it contains entries of the form $(cp, rect, cp.IntUnidi, cp.num)$, where $cp.num$ is the total number of objects stored in the subtree rooted at R. Here, $cp.IntUnidi$ is the identifier for the union and intersection of all text descriptions in the entries of the child node. Figure 4 illustrates the MIUR-tree for $U = \{u_1, u_2, \ldots, u_7\}$ of Figure 2b, where the MBRs are constructed according to the IR-tree (not shown in figure), and the table shows the text vectors of the nodes for the users presented in Table 3.

Detailed steps. The root of the MIUR-tree is essentially the same as the super-user u_s. Lines 1.4-1.25 of Algorithm 1 is executed for the root in the same manner as u_s, and we obtain the priority queues of objects, LO and RO. The k-th best lower bound score $RS_k(u_s)$ is also obtained for the root. For each $\ell \in L$, a list LU_ℓ is maintained, but unlike Algorithm 3, LU_ℓ may now contain user nodes. In each iteration, the location $max\ell$ is selected with the maximum $|LU_\ell|$. If there is a user node in a LU_ℓ, the number of actual users stored in that subtree is used to compute the number of users in LU_ℓ. The following steps are executed to access the MIUR-tree-

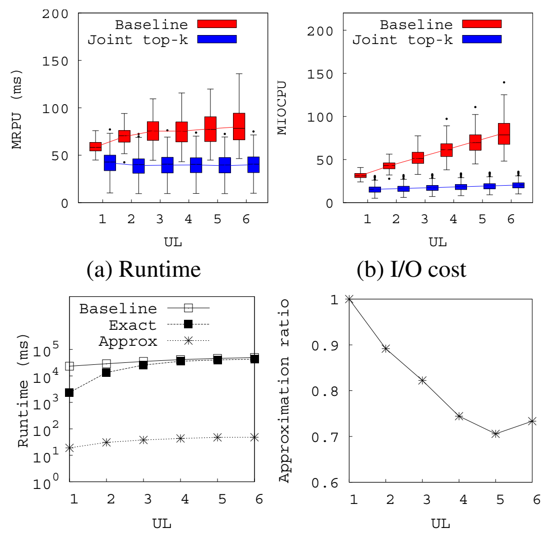
1. If there is any non-leaf node in $LU_{max\ell}$, we dequeue the non-leaf node $EU \in LU_{max\ell}$ with the maximum number of users stored in the subtree.
   (i) The MIUR-tree is accessed to retrieve the elements of EU. For EU, the INDIVIDUAL_TOPK$(EU, parent(EU), LO, RO)$ of Algorithm 2 is executed. Thus, the score $RS_k(eu)$ of each user $eu \in EU$ is updated. Each eu also inherits the priority queues LO and RO with updated scores. The upper bound scores of $max\ell$ are computed for each $eu \in EU$, and inserted in $LU_{max\ell}$ if eu can be a BRSTkNN. The $max\ell$ is enqueued in Q_L with the updated $LU_{max\ell}$.
   (ii) For each $\ell \in L$, if $EU \in LU_\ell$, the list is updated with the users $eu \in EU$ based on the corresponding upper bound scores. The priority queue Q_L is also updated. In this way, we need to access a node of the MIUR-tree at most once.



(a) Runtime (b) I/O cost

(c) Runtime (d) Approximation ratio

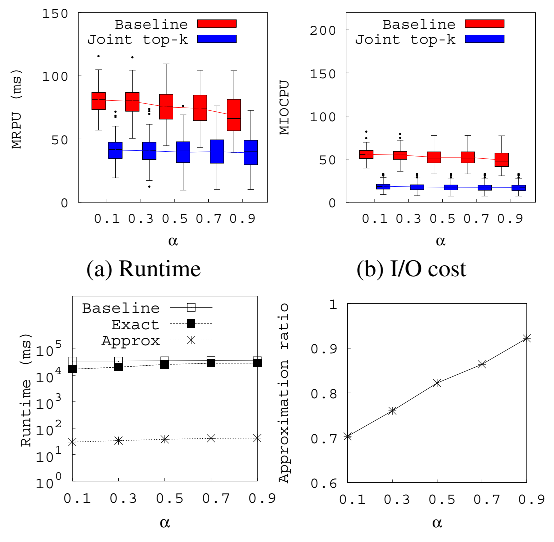
Figure 5: Effect of varying k



(a) Runtime (b) I/O cost

(c) Runtime (d) Approximation ratio

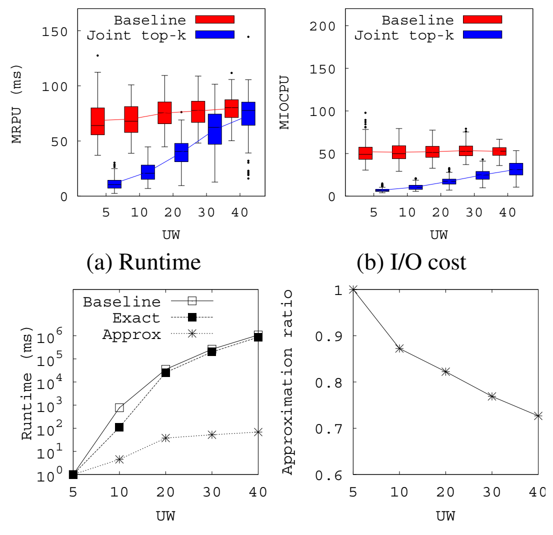
Figure 7: Effect of varying UL



(a) Runtime (b) I/O cost

(c) Runtime (d) Approximation ratio

Figure 6: Effect of varying α



(a) Runtime (b) I/O cost

(c) Runtime (d) Approximation ratio

Figure 8: Effect of varying UW

2. Otherwise, the rest of the algorithm to find the best tuple of candidate location and keyword set $\langle \ell, W' \rangle_{best}$ with the maximum number of BRSTkNNs is same as Algorithm 3.

In this best-first method, the users that are in the BRSTkNN of the most promising candidate, are accessed first. In addition, the top-k objects are not computed for the users that are not necessary to determine the best candidates.

8. EXPERIMENTAL EVALUATION

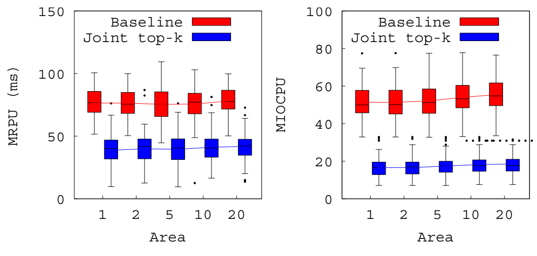
In this section, we report the experimental evaluation of the algorithms for processing the query with two real datasets, and compare them with the baseline approach.

Datasets and user generation. All experiments are conducted on two real datasets, namely, (i)Yahoo I3 Flickr dataset [1], and (ii) Yelp dataset [2]. For the Flickr dataset, a total of 1 million images that are geo-tagged and contain at least one user specified tag were extracted from the collection. The locations and tags are used as the location and keywords of the objects. Three additional datasets of 2 million, 4 million, and 8 million were extracted from the collection to evaluate scalability by varying the size of the set of objects, O. The Yelp dataset contains locations of the businesses, business attributes, and users' reviews. The attributes and reviews of each business are combined as the text description of that business. Table 4 lists the properties of the datasets.

For the Flickr dataset, we generate the set of the users using the dataset as follows. First, an area of a fixed size (here, 5 latitude $\times 5$ longitude) is chosen and a pre-defined number ($|U|$) of objects O_u in that area are taken randomly. The locations of the objects are used as the locations of the users. Then, UW keywords are randomly selected from O_u as the set of the user keywords. These keywords are distributed among the users such that each user has

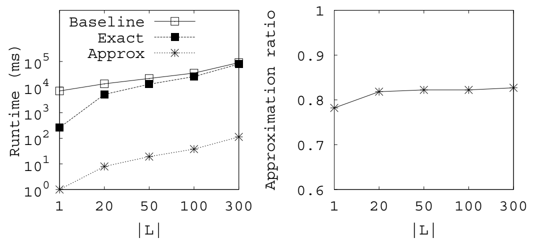
[1]http://webscope.sandbox.yahoo.com/catalog.php?datatype=i&did=67

[2]http://www.yelp.com.au/dataset_challenge



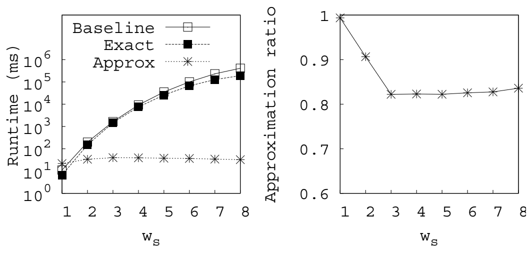
(a) Runtime (b) I/O cost

Figure 9: Effect of varying *Area*



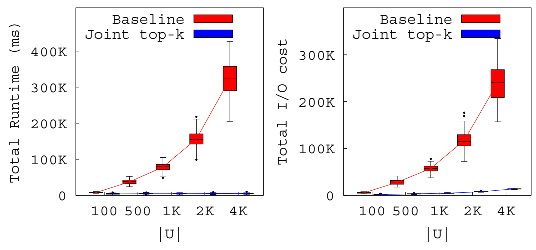
(a) Runtime (b) Approximation ratio

Figure 10: Effect of varying $|L|$

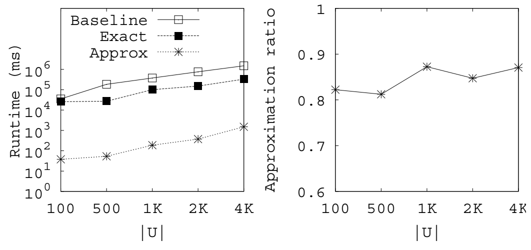


(a) Runtime (b) Approximation ratio

Figure 11: Effect of varying w_s



(a) Runtime (b) I/O cost



(c) Runtime (d) Approximation ratio

Figure 12: Effect of varying $|U|$

$|UL|$ number of keywords following the same distribution of keywords of O_u. In this work, we generate 100 such sets of users and report the average performance. The set of keywords UW is used as the set of candidate keywords.

For the Yelp dataset, the location of a user is taken as the centroid of the locations of the businesses she reviewed. The text description of a user is the collection of the reviews posted by that user. We pick $|U|$ number of such users. The keywords $|UW|$ are chosen from the combined text descriptions of the users, and the keywords are distributed is the same manner as mentioned above.

Setup. All indexes and algorithms are implemented in Java. The experiments were ran on a 24 core Intel Xeon $E5-2630$ running at 2.3 GHz using 256 GB of RAM, and 1TB 6G SAS 7.2K rpm SFF (2.5-inch) SC Midline disk drives. The Java Virtual Machine Heap size was set to 4 GB. All index structures are disk resident, and the page size was fixed at 4 kB.

As multiple layers of cache exist between a Java application and the physical disk, we report simulated I/O costs in the experiments instead of physical disk I/Os. The number of simulated I/Os is increased by 1 when a node of a tree is visited. When an inverted file is loaded, the number of simulated I/Os is increased by the number of blocks (4 kB per block) for storing the list. In the experiments, the performance is evaluated using cold queries. Unless stated otherwise, Flickr dataset is used.

8.1 Performance evaluation

The performance evaluation of our proposed approaches consists of two components:

- The top-k objects of the users are computed. We compare the joint top-k processing with the baseline, which computes the top-k objects of the users individually.
- Then, the best combination of the location and the keywords is selected from the given set of candidates. We compare the performances of an exact and an approximate method.

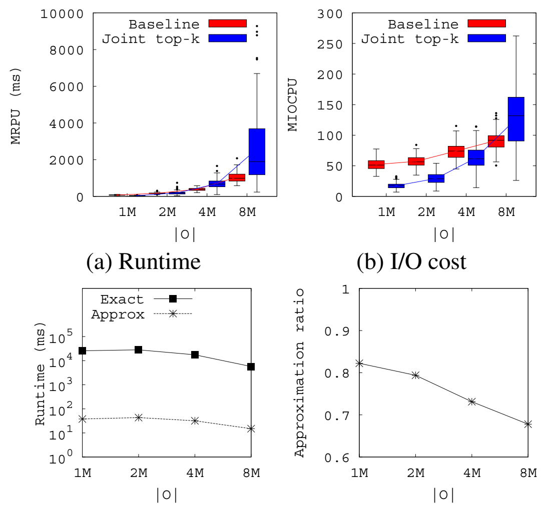
In this section, we evaluate the performances by varying several parameters. The parameter ranges are listed in Table 5 where the values in bold represent the default values. We also compare the performances for three different text relevance measures, namely, the TF-IDF, Language Model (LM), and the Keyword Overlap (KO).

In all experiments, we use the default settings and vary a single parameter to study the impact on: (i) the mean runtime per user (MRPU) to compute the top-k objects; (ii) the mean I/O cost per users (MIOCPU) to compute the top-k objects; (iii) the total runtime of selecting the best candidate; and (iv) the approximation ratio of the approximate and the exact method of MaxBRSTkNN. This value is the ratio between the number of BRSTkNNs for the best candidate returned by the approximate method, and the number of BRSTkNNs for the best candidate returned by the exact method.

Varying *k*. Figure 5 shows the experiments for varying k. Since the joint top-k processing (J) employs several pruning strategies, and avoids visiting any page multiple times, the costs are significantly lower than the baseline (B). For keyword overlap (KO), several objects may have the same number of keyword as the user. Therefore, more objects must to be considered when computing the top-k objects, resulting into a higher cost than the other text similarity measures considered here.

Note that we tested all three of the similarity metrics with several different parameter settings, and the overall trends were similar to those shown in Figure 5. Since the costs in the baseline are the lowest when using the language model, we use only this similarity metric for the remainder of the experiments.

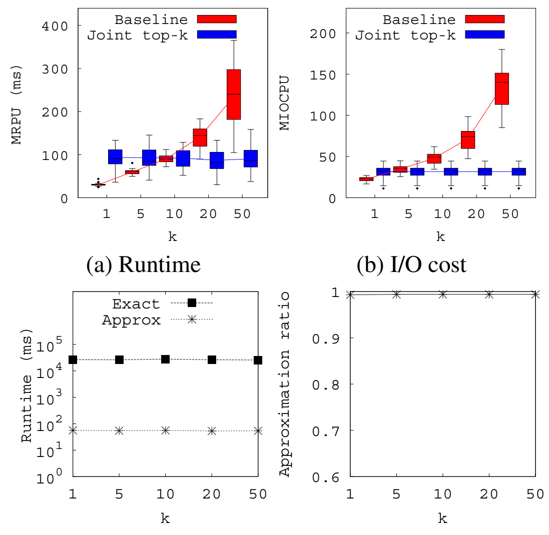
The runtime of the baseline for selecting the best candidate does not change for k as it exhaustively computes all candidate combinations. The approximate method (A) selects the candidate combination of keywords greedily, and therefore requires 3 orders of magnitude less computational time than the exact method. The accuracy of the approximation algorithm increases with k as more candidates become eligible to be included in the answer. As the



(a) Runtime (b) I/O cost

(c) Runtime (d) Approximation ratio

Figure 13: Effect of varying $|O|$



(a) Runtime (b) I/O cost

(c) Runtime (d) Approximation ratio

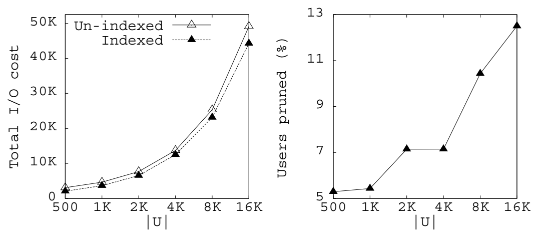
Figure 14: Effect of varying k on Yelp dataset

weights for each candidate keyword are the same in the keyword overlap measure, the approximate algorithm is able to track the exact method closely in terms of accuracy.

Varying α. Figure 6 shows the results when varying α. A higher value indicates more preference to spatial similarity. As the IR-tree groups the objects based on spatial similarity, a higher α leads to lower cost in the baseline method. In the joint top-k approach, as the MBR of the users' locations, and the union of the users' keywords remain the same, the cost remains almost constant. The accuracy of the approximate method increases with the increase of α as more candidate locations become eligible to become an answer.

Varying *UL*. We now vary the number of keywords per user, and present the effect on performance in Figure 7. The cost of the baseline increases proportionally with the increase of UL, as more objects become relevant to these users. In contrast, the I/O cost of our proposed joint top-k algorithm remains almost constant as a node is retrieved at most once, and processed for all the users concurrently.

Here, the number of BRSTkNNs increase with the increase of UL for both exact and approximate methods, but they do not in-



(a) Total I/O cost (b) Users pruned (%)

Figure 15: Effect of varying $|U|$

Table 4: Description of dataset

Property	Flickr	Yelp
Total objects	1,000,000	61,185
Total unique terms	166,317	266,869
Avg unique terms per object	6.9	398.7
Total terms in dataset	6,936,385	77,838,026

crease in the same rate. As a result, a local minima appears in the graph of the approximation ratio ($UL = 5$). When the rate of increase in the number of BRSTkNNs decreases for the exact method, the approximation starts to improve.

Varying *UW*. Figure 8 shows the effect on performance when varying the total number UW of unique keywords of the users. Here, a lower value indicates that the users share more keywords. As the joint top-k method exploits the shared I/Os among users, it outperforms the baseline where the benefit is higher for higher overlap. As the set of keywords UW is also the set of candidate keywords, the runtime of candidate selection increases for both the exact and the approximate methods. As UW increases, the possible combinations of the candidate keywords also increases. Therefore the accuracy of the approximate method is very high for the lower values of UW, and decreases gradually as UW increases.

Varying *Area*. Figure 9 shows the results when varying the MBR of the users by latitude and longitude (Area). A higher value indicates that the locations of the users are more sparse. In this setting, as the union of the users' keywords remain the same, the joint top-k processing benefits from these shared I/Os even when the size of the MBR increases. The approximate method follows the exact method better when the users are sparse. The graphs of the candidate selection step is not shown due to page limitations.

Varying $|L|$. Figure 10 shows the results when varying the number of candidate locations $|L|$. As the top-k processing of the users do not depend on $|L|$, we have only shown the performance of the exact and the approximate methods. The methods differ in the manner of selecting the candidate keywords, and the runtime increases proportionally with the increase of $|L|$ in both methods. The accuracy of the approximation increases slightly for a higher value of $|L|$, as more candidate locations become potential results.

Varying w_s. The performances when varying w_s is shown in Figure 11. As the number of keyword combinations increases with w_s, the runtime of the baseline and the exact methods also increase. The number of BRSTkNNs increases rapidly as $|w_s|$ increases, and the accuracy of the approximate method drops as well. When the rate of increase of the number of BRSTkNNs start to level off (here, for $w_s > 3$), the accuracy of the approximation gradually improves.

Varying $|U|$. In Figure 12, we vary the number of users and show the performance. As the baseline computes the top-k objects of the users individually, the cost increases rapidly with $|U|$. The joint top-k approach considers the MBR, and the union-intersection of

Table 5: Parameters

Parameter	**Range**
k	$5, \mathbf{10}, 20, 50, 100$
α	$0.1, 0.3, \mathbf{0.5}, 0.7, 0.9$
No. of keywords per user, UL	$1, 2, \mathbf{3}, 4, 5, 6$
No. of total unique keywords of users, UW	$5, 10, \mathbf{20}, 30, 40$
Users' MBR as latitude $\times$ longitude, $Area$	$1, 2, \mathbf{5}, 10, 20$
No. of candidate locations, $\|L\|$	$1, 20, 50, \mathbf{100}, 300$
w_s	$1, 2, 3, 4, \mathbf{5}, 6, 7, 8$
No. of users, $\|U\|$	$\mathbf{100}, 500, 1K, 2K, 4K$
No. of objects, $\|O\|$	$\mathbf{1M}, 2M, 4M, 8M$

the users' keywords, therefore the costs do not vary much. This experiment shows that joint top-k processing is in fact scalable.

Varying $|O|$. We vary the number of objects and show the performance in Figure 13. As $|O|$ increases, we need to consider more objects that can be a top-k object of at least one of the users. Therefore, the cost increases in both methods. As $|O|$ increases, the relevance score of the k-th ranked object of a user is likely to improve. Therefore, more candidates can be pruned for higher values of $|O|$.

Experiments on Yelp dataset. We have conducted the same experiments as mentioned above on the Yelp dataset. Due to page limitations, we report our results when varying k in the Yelp dataset in Figure 14. All of our experimental results were consistent across both datasets.

Performance with a user index. Here, we index the users with the MIUR-tree and compare the performance with the non-indexed case. When the users are indexed, we can avoid computing the top-k objects for some users. This pruning capacity is shown using the metric "Users pruned (%)", i.e., the percentage of the users for which we do not compute the top-k objects to answer MaxBRSTkNN query. Figure 15 shows the performance when the number of the users is varied. For the indexed users, the total I/O cost indicates the combined I/O of the MIR-tree of the objects and the MIUR-tree of the users. As shown, the performance of the indexed users approach is better as the top-k objects are not computed for the pruned users. The percentage of the users pruned are between 5–12.5%, where the pruning capacity increases with the increase of $|U|$.

9. CONCLUSION

The paper introduced and explored a novel query type, the *Maximized Bichromatic Reverse Spatial Textual k Nearest Neighbor (*MaxBRSTkNN*)* query which finds an optimal location, and a set of keywords that maximizes the size of the bichromatic reverse spatial textual k nearest neighbors. The problem has several real-life applications. We proved that the MaxBRSTkNN problem is NP-hard, and proposed an approximate and an exact method to answer the query. We also provided an efficient method to compute the top-k objects jointly, which is of independent interest and improves the overall performance of the query processing. From the experiments we have shown that the approximate algorithm is around 2-3 magnitude faster than the exact method.

Acknowledgments. This work was supported by the ARC *Discovery Projects* Scheme (DP140101587). Shane Culpepper is the recipient of an ARC DECRA Research Fellowship (DE140100275). Farhana Choudhury is the recipient of a NICTA scholarship. We thank Chen et al. [1] for providing the IR-tree implementation. We thank the anonymous reviewers for their comments. An extended version of the paper is available in the first author's personal website that includes additional graphs and explanations as recommended by one of the reviewers.

10. REFERENCES

[1] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: an experimental evaluation. In *VLDB*, pages 217–228, 2013.

[2] L. Chen-Yi, K. Jia-Ling, and A. P. Chen. Determining k-most demanding products with maximum expected number of total customers. *TKDE*, 25(8):1732–1747, 2013.

[3] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[4] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[5] D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

[6] J. Huang, Z. Wen, J. Qi, R. Zhang, J. Chen, and Z. He. Top-k most influential locations selection. In *CIKM*, pages 2377–2380, 2011.

[7] J. J. Cardinal and S. Langerman. Min-max-min geometric facility location problems. In *EWCG*, pages 149–152, 2006.

[8] Q. Jianzhong, Z. Rui, L. Kulik, D. Lin, and X. Yuan. The min-dist location selection query. In *ICDE*, pages 366–377, 2012.

[9] J.-L. Koh, C.-Y. Lin, and A. P. Chen. Finding k most favorite products based on reverse top-t queries. *PVLDB*, 23(4):541–564, 2014.

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

[11] H. Lin, F. Chen, Y. Gao, and D. Lu. OptRegion: Finding optimal region for bichromatic reverse nearest neighbors. In *DASFAA*, pages 146–160, 2013.

[12] Y. Liu, R.-W. Wong, K. Wang, Z. Li, C. Chen, and Z. Chen. A new approach for maximizing bichromatic reverse nearest neighbor search. *Knowledge and Information Systems*, 36(1):23–58, 2013.

[13] J. Lu, Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD*, pages 349–360, 2011.

[14] Y. Lu, J. Lu, G. Cong, W. Wu, and C. Shahabi. Efficient algorithms and cost models for reverse spatial-keyword k-nearest neighbor search. *ACM Trans. Database Syst.*, 39(2):1–46, 2014.

[15] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[16] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *SSTD*, pages 205–222, 2011.

[17] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Norvag. Monochromatic and bichromatic reverse top-k queries. *Knowledge and Data Engineering, IEEE Transactions on*, 23(8):1215–1229, 2011.

[18] A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis. Identifying the most influential data objects with reverse top-k queries. *PVLDB*, 3(1-2):364–372, 2010.

[19] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng. Creating competitive products. *Proc. VLDB Endow.*, 2(1):898–909, 2009.

[20] R. C.-W. Wong, M. T. Özsu, A. W.-C. Fu, P. S. Yu, L. Liu, and Y. Liu. Maximizing bichromatic reverse nearest neighbor for lp-norm in two and three-dimensional spaces. *PVLDB*, 20(6):893–919, 2011.

[21] R. C.-W. Wong, M. T. Özsu, P. S. Yu, A. W.-C. Fu, and L. Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *PVLDB*, 2(1):1126–1137, 2009.

[22] D. Yan, R. C.-W. Wong, and W. Ng. Efficient methods for finding influential locations with adaptive grids. In *CIKM*, pages 1475–1484, 2011.

[23] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179–214, 2004.

[24] D. Zhang, Y. Du, T. Xia, and Y. Tao. Progressive computation of the min-dist optimal-location query. In *VLDB*, pages 643–654, 2006.

[25] L. Zhisheng, K. C. K. Lee, Z. Baihua, L. Wang-Chien, L. Dik Lun, and W. Xufa. IR-tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, 2011.

[26] Z. Zhou, W. Wu, X. Li, M. L. Lee, and W. Hsu. MaxFirst for MaxBRkNN. In *ICDE*, pages 828–839, 2011.