Level-aware Collective Spatial Keyword Queries

Pengfei Zhang[a], Huaizhong Lin[a,*], Bin Yao[b], Dongming Lu[a]

[a] College of Computer Science and Technology, Zhejiang University, Hangzhou, China.

[b] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China.

Abstract

The collective spatial keyword query (CoSKQ), which takes a location and a set of keywords as arguments, finds a group of objects that collectively satisfy the query and achieve the smallest cost. However, few studies concern the *keyword level* (e.g., the level of hotels), which is of critical importance for decision support. Motivated by this, we study a novel query paradigm, namely *Level-aware Collective Spatial Keyword* (LCSK) query. The LCSK query asks for a group of objects that cover the query keywords collectively with a threshold constraint and minimize the cost function, which takes into account both the cost of objects and the spatial distance. In our settings, each keyword that appears in the textual description of objects is associated with a level for capturing the feature of keyword.

We prove the LCSK query is NP-hard, and devise exact algorithm as well as approximate algorithm with provable approximation bound to this problem. The proposed exact algorithm, namely MergeList, explores the candidate space progressively with several pruning strategies, which is based on the keyword hash table index structure. Unfortunately, this approach is not scalable to large datasets. We thus develop an approximate algorithm called MaxMargin. It finds the answer by traversing the proposed LIR-tree in the best-first fashion. Moreover, two optimizing strategies are used to improve the query performance. The experiments on real and synthetic datasets verify that the proposed approximate algorithm runs much faster than the competitor with desired accuracy.

Keywords: Collective spatial keyword query, Keyword level, Branch and bound strategy, Triggered update strategy

1. Introduction

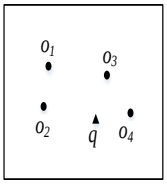
Spatial database has been studied for decades as it supports many applications from people's daily life to scientific research [2, 3, 10, 23, 26, 34, 42, 47]. Recently, the keyword search has been combined with spatial queries to enhance location-based services such as Baidu Lvyou and Google Earth. Previous works on spatial keyword queries can be roughly classified into two categories based on the answer granularity: (1) some proposals find individual objects. Typically, given a location and a set of keywords as arguments, this type of query [14, 15, 17, 39] retrieves individual objects that each can cover all query keywords, and (2) others ask for a group of objects. In a wide spectrum of applications, whereas multiple objects are required to satisfy the user's needs (expressed by keywords) collectively. Toward this goal, mCK [51, 52], CoSKQ [7, 35], BKC [18] and SGK [6] are investigated. To the best of our knowledge, few studies consider the

*Corresponding author.

Email addresses: zpf_2013zb@zju.edu.cn (Pengfei Zhang[a]), linhz@zju.edu.cn (Huaizhong Lin[a,]), yaobin@cs.sjtu.edu.cn (Bin Yao[b]), ldm@zju.edu.cn (Dongming Lu[a])

[1] This manuscript is the authors' original work and has not been published nor has it been submitted simultaneously elsewhere.

[2] All authors have checked the manuscript and have agreed to the submission.



object	keywords	level vector	cost	distance
o_1	t_1, t_2, t_3	4, 5, 1	0.1	10
o_2	t_1, t_2, t_4	5, 4, 1	0.1	8
o_3	t_1, t_2, t_3, t_4	3, 2, 5, 5	0.15	6
o_4	t_1, t_2, t_3, t_5	2, 3, 5, 5	0.15	7

Fig. 1: Example of the LCSK query

keyword level. In real applications, we can use the keyword level to capture the level of tourist attractions, hotels or the rescue ability of equipments, which is increasingly important for users to make decisions.

In this work, we study a novel query paradigm called Level-aware Collective Spatial Keyword (LCSK) query. We enhance the collective spatial keyword queries (CoSKQ) from the following two aspects. First, we introduce the *level vector* for the objects in the database O. Each object $o \in O$ is associated with an integer level vector, denoted by $o.\nu$. The ith element of a level vector, i.e., $o.\nu^i$, represents the level of ith keyword in $o.\omega$, i.e., $o.\omega^i$. We denote by $o.\omega$ the associated keywords with o. Second, we introduce a normalized *weight vector* into the query definition for capturing the user-specified weights assigned to different levels. Similar to [12], we define our cost function, namely *cost distance*, as a combination of the spatial distance and the cost of objects (will be explained later). The LCSK query has numerous real applications such as resource scheduling and emergency rescue. Next, we present an example of the emergency rescue task.

Example 1. As illustrated in Fig. 1, we assume that the query point q is an earthquake point. There are four rescue teams o_1, o_2, o_3 and o_4 having necessary rescue equipments, namely $t_1, t_2, ..., t_5$. The *level vector* captures the level of equipments that are associated with teams, which can be used to measure the rescue ability. Usually, the higher the level, the higher the rescue ability. The *cost* denotes the overhead of performing the rescue by the corresponding team, and *distance* denotes the Euclidean distance between q and the rescue team. In such a scenario, we aim to find multiple teams that can together achieve the required rescue ability and have the smallest cost.

To address the above problem, we issue a query $q = (\ell, \omega, W, \theta)$, where ℓ denotes the location of q and $\omega = \{t_1, t_2\}$ captures the required rescue equipments. The normalized weight vector $W = (0.1, 0.15, 0.2, 0.25, 0.3)$ indicates the rescue ability of equipments with different levels. As an example, the level of t_1 w.r.t. o_1 is 4, and thus the corresponding rescue ability is $W[4] = 0.25$. Then, $\theta = 0.5$ denotes the desired rescue ability for each required equipment. That is, a group of teams whose rescue abilities are not less than 0.5 for each required equipment are called for. In this case, we deliver the group $\{o_1, o_2\}$ as the answer, because it offers the desired rescue ability for each required equipment and has the smallest cost distance. Specifically, the rescue ability of t_1 and t_2 contributed by $\{o_1, o_2\}$ are $W[4] + W[5] = 0.55$ and $W[5] + W[4] = 0.55$, which are larger than the given threshold 0.5. Moreover, the cost distance of $\{o_1, o_2\}$ is $0.1 * 10 + 0.1 * 8 = 1.8$.

More formally, given a spatial database O, and an LCSK query $q = (\ell, \omega, W, \theta)$, where ℓ is the query location and ω is the set of query keywords. W is a normalized weight vector and θ is a threshold. The LCSK query is to retrieve a group G of objects such that satisfy the following two conditions:

- $\forall t \in q.\omega$, the coverage weight of t by G is not less than $q.\theta$;
- G has the smallest cost distance among those groups that meet the above condition.

We prove the LCSK query is NP-hard by the reduction from the weighted set cover (WSC) problem. We devise both the exact algorithm and approximate algorithm to this problem. The proposed exact algorithm, namely MergeList, performs the query by searching the candidate space progressively, which is based on the keyword hash table index structure. Specifically, the candidate space contains all promising answers, and we use several strategies to prune the candidate space. Though equipped with several pruning strategies, MergeList is not scalable to large datasets due to the complexity of our problem. We thus develop an approximate algorithm called MaxMargin with provable approximation bound. MaxMargin finds the answer by traversing the LIR-tree in the best-first fashion. In particular, the LIR-tree augments each inverted file of IR-tree with additional information, i.e., the level of keywords and the cost of objects. Two effective

optimizing strategies, namely the *branch and bound strategy* (BBS) and the *triggered update strategy* (TUS) are proposed to further improve the performance of MaxMargin.

To summarize, we make the following contributions:

- We formally define the LCSK query, which is to find a group of objects such that collectively cover the query keywords with a threshold constraint and have the smallest cost distance. We then theoretically prove this problem is NP-hard.
- We develop an exact algorithm called MergeList on the top of keyword hash table index structure. Furthermore, we propose the LIR-tree, which is extended from IR-tree. Based on the LIR-tree, we devise an approximate algorithm called MaxMargin that finds the answer using the best-first strategy. To further facilitate the query processing, two optimizing strategies, namely BBS and TUS, are adopted by MaxMargin.
- We conduct comprehensive experiments on real and synthetic datasets to verify the performance of our proposed algorithms.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 formally defines the problem and proves the LCSK query is NP-hard. We elaborate the exact algorithm in Section 4 and the approximate algorithm in Section 5, respectively. Finally, we report on empirical studies in Section 6 and offer conclusions and research directions in Section 7.

2. Related Work

2.1. Conventional Spatial Keyword Queries

The conventional spatial keyword queries [14, 17] take a location and a set of keywords as arguments, and ask for the objects that each can cover all the query keywords. There are lots of efforts on the conventional spatial keyword queries. We proceed to review them.

Combining with top-k queries. The top-k spatial keyword queries return k objects with the highest ranking scores measured by a ranking function, which considers the spatial proximity and the textual relevance. To answer the query efficiently, various index structures are proposed such as IR-tree [14, 33], SKI [9], S2I [37, 38], IL-Quadtree [50]. Cong et al. [15] use the B^{ck}-tree to facilitate the query processing for the trajectory data. Wu et al. [46] handle the joint top-k spatial keyword queries with the W-IR-Tree, which utilizes keyword partitioning and inverted bitmaps to organize the objects. Zhang et al. [53] show that I^3 outperforms IR-tree and S2I. Specifically, I^3 adopts the quadtree to hierarchically partition the data space into cells. Gao et al. [19] study the reverse top-k boolean spatial keyword queries and use the count tree to maintain the shortest paths. Most recently, Chen et al. [11] develop an index-based method to answer the why not top-k spatial keyword queries, which enable to modify the query for retrieving desired results. Wu et al. [45] study the authentication of moving top-k spatial keyword queries using the MIR-tree, which modifies the IR-tree by embedding a series of digests in each node of the tree.

Combining with nearest neighbor queries. The nearest neighbor (NN) queries [41, 49] are to retrieve the nearest object to one or a group of query points from the underlying database. By considering keywords, the spatial keyword nearest neighbor queries return the closest object to the query point among those objects that cover the query keywords. Recently, several variations are explored. To address the false hit problem of IR2-tree [17], Tao et al. [43] propose the SI-index. It is actually a compressed version of inverted list using the gap-keeping technique. Lu et al. [36] study the RSTkNN query, finding objects that take a specific query object as one of their k most spatial-textual similar objects. Jiang et al. [25] study the k nearest keyword (k-NK) query in the context of large networks. They develop memory-based and disk-based exact methods using the label technique.

Combining with route queries. The conventional route queries [29] search the shortest route that starts at location s, passes through as least one object from each category in C and ends at t. Approximate algorithms are proposed in [29] for various applications. Lee et al. [27] propose an initialization method for the robot path planning based on the directed acyclic graph. Yao et al. [48] investigate the multi-approximate-keyword routing (MARK) query. MARK asks for the shortest route that covers at least one matching object per

keyword with the similarity greater than the given threshold. The problem of keyword-aware optimal route (KOR) search is studied in [5]. It retrieves the route that covers a set of given keywords, a specific budget constraint is satisfied, and an objective score of the route is optimal. Since the query is NP-hard, three approximate algorithms are developed. In the subsequent work [4], they present the corresponding system. Cao et al. [8] are to find a length-constraint region (expressed by a connected subgraph of road networks) that best matches the query keywords. Approximate algorithms are developed to answer it due to the inherent complexity of this problem.

It is clear that all these works are different from ours. Specifically, aforementioned works only return individual objects for satisfying users' needs. However, our query aims to find a group of objects that satisfy users' needs collectively. Thus, their methods cannot be used to solve our problem.

2.2. Collective Spatial Keyword Queries

The m-closed keywords (mCK) queries are studied in [51, 52]. Given a query point constituted by a location and a set of keywords, mCK asks for a group of objects that together cover all keywords and have the minimum diameter. The diameter is defined to be the largest distance between any pair of objects in the group. Zhang et al. [51] answer the mCK query by traversing down the bR^*-tree in a depth-first manner. In addition, the priori-based search strategies are exploited to shrink the search space. Later, Zhang et al. [52] investigate the mCK query with the web data. The bottom-up strategy is developed to find a candidate answer, which prunes unnecessary accesses significantly. To further facilitate the query processing, Guo et al. [21] answer the mCK query approximately by searching the smallest diameter circle in which the final answer locates.

Cao et al. [7] study the CoSKQ problem. First, they aim to minimize the sum of distances between the retrieved group of objects and query point. They develop a greedy method that finds the result by answering a sequence of partial queries progressively. In addition, an exact algorithm is proposed by exploiting the dynamic programming. Second, they investigate the cost function that considers both the inter-object distance and the distance between the object in the retrieved group and query point. A simple approximate algorithm is first proposed to address this problem. It retrieves the nearest object for each query keyword, and then merges them as the result set. Based on this method, another approximate method is developed by refining the retrieved group progressively. Finally, an exact algorithm is presented. Specifically, this exact algorithm first invokes the second approximate algorithm for deriving an upper bound, and then shrinks the search space with pruning strategies. Later, they extend this work by exploring other cost functions in [6]. Long et al. [35] study CoSKQ on two cost functions, namely the maximum sum cost and the diameter cost. With the distance owner-driven approach, they answer the query exactly and approximately.

Recently, Gao et al. [20] study CoSKQ on road networks based on the Connectivity-Clustered Access Method (CCAM) index. Skovsgaard et al. [40] propose to find top-k disjoint groups of objects while considering the group density, distance to the query, and relevance to the query keywords. Efficient algorithms are presented based on the Group Extended R-Tree, which is extended from the R-tree with each node including compassed histograms. Deng et al. [18] study the best keyword cover (BKC) query, which considers the keyword rating. To answer this query, keyword nearest neighbor expansion algorithm returns the local best solution with the highest score as the answer. Wang et al. [44] propose to explore the spatial keyword query over streaming data with the AP-tree. These works distinguish from ours, since they do not consider the level and have different query goals with ours.

3. Problem Statement

Let $O = \{o_1, ..., o_n\}$ be a spatial database. Each object $o \in O$ is associated with a location $o.\ell$, a set of keywords $o.\omega$ and a level vector $o.\nu$ including $|o.\omega|$ elements. As mentioned, the ith element of $o.\nu$, i.e., $o.\nu^i$, represents the level of corresponding keyword in $o.\omega$, i.e., $o.\omega^i$. We denote *keyword level* by *level* and consider it as a positive integer hereafter. As with [6], we assume that each object o has a positive cost, namely $cost(o)$, which plays a critical role in decision support. Consider, for example, $cost(o)$ can capture the user ratings of products. Intuitively, the higher the rating, the more people will compete for such products, which incurs higher costs.

Table 1: Summary of the notations

Notation	Explanation
q	an LCSK query of the form:$(\ell, \omega, W, \theta)$
O, o	the underlying database, an object in O of the form: (ℓ, ω, ν)
RO_q	relevant objects to q in O
G	the answer (i.e., a group of objects) to q
$\|S\|$	the cardinality of S
$cost(o)$	the cost of an object o
$cd(o, q)$	the cost distance between o and q
$cw(o, t)$	coverage weight of t by o
$cov(G, q)$	coverage weight of q by G
KHT	keyword hash table index structure
LIR-tree	index structure extended from the IR-tree
$cr(e, q), cr$	contribution ratio of the entry e to q, the abbreviation of contribution ratio
$dcr_q^r(e)$	dynamic contribution ratio of the entry e to q when $\|G\| = r$
dcr	the abbreviation of dynamic contribution ratio

Definition 1. (Cost Distance) Given a query q and an object $o \in O$, we define the cost distance between o and q as:

$$cd(o, q) = cost(o) \cdot dist(o, q) \tag{1}$$

In Equation (1), $dist(o, q)$ represents the Euclidean distance between o and q. The cost distance is more realistic compared with the cost functions in [7, 35], since the objects are always associated with the *cost* for highlighting some features of objects in real applications.

Definition 2. (Coverage weight) Given a keyword t, an object $o \in O$ and a query $q = (\ell, \omega, W, \theta)$. We denote by $o.\nu_t$ the corresponding level of t in $o.\nu$. We define the coverage weight of t by o as:

$$cw(o, t) = W[o.\nu_t] \tag{2}$$

Different from CoSKQ [7, 35] in which t is either covered by an object o or not, in this work we consider the coverage weight. Note that if $t \notin o.\omega$, we set $cw(o, t)$ to 0. We use the notations $cov(o, q) = \sum_{t \in q.\omega} cw(o, t)$ and $cov(G, t) = \sum_{o \in G} cw(o, t)$ to denote the coverage weight of q by o and the coverage weight of t by G, respectively. In Example 1, we know that $cov(o_1, q) = cw(o_1, t_1) + cw(o_1, t_2) = W[4] + W[5] = 0.25 + 0.3 = 0.55$. In addition, if $cw(o, t)$ is greater than $q.\theta$ then we set it to $q.\theta$.

Definition 3. (Level-aware Collective Spatial Keyword Query) Given an LCSK query $q = (\ell, \omega, W, \theta)$, where ℓ is the query location and ω is the set of query keywords. W is a normalized weight vector and θ is a threshold. The answer of the query q is a group G of objects that satisfy the following two conditions:

- $\forall t \in q.\omega$, it holds that $cov(G, t) \geq q.\theta$ (threshold constraint);
- $\arg\min_{G} \sum_{o \in G} cd(o, q)$.

Given a query q, we say that an object o is *relevant* to q if $\exists t \in q.\omega$, and it holds that $t \in o.\omega$. We denote by RO_q all the relevant objects to q in the database O, that is, $RO_q = \{o | o \in O \wedge (\exists t \in q.\omega \wedge t \in o.\omega)\}$. We then only need to consider RO_q for a specific query q. We say that G is a *feasible solution* to q if G satisfies the threshold constraint in Definition 3, i.e., $\forall t \in q.\omega$, we have $cov(G, t) \geq q.\theta$. In other words, the LCSK query returns the *feasible solution* with the smallest cost distance as the answer. Given a query q, when there are multiple optimal groups of objects, we choose one group randomly. For ease of reference, Table 1 summarizes the notations used widely in this paper.

Table 2: Example of KHT entries

entry	keyword	object list	first index	second index
e_1	t_1	o_1, o_3, o_7, o_{10}	5	0
e_2	t_2	o_3, o_9	6	1
e_3	t_3	o_1, o_7	6	2
e_4	t_4	o_2, o_4, o_5	6	1
e_5	t_5	o_1, o_3, o_6	1	0
e_6	t_6	o_2, o_7	2	0
e_7	t_7	o_5, o_6	5	1
e_8	t_8	o_5, o_{10}	0	0
e_9	t_9	o_8, o_{10}	4	0

Theorem 1. *The LCSK query is NP-hard.*

PROOF. *To prove the LCSK query is NP-hard, we reduce the WSC problem to it. Typically, an instance of the WSC problem is of the form* $< U, S, C >$*, where* $U = \{1, 2, 3, ..., n\}$ *consists of* n *elements and* $S = \{S_1, S_2, S_3, ..., S_m\}$ *consists of a family of sets, and* $S_i \subseteq U \wedge \cup S_i = U$*. Each* $S_i \subseteq U$ *is associated with a positive cost* $c_i \in C$ *indicating the weight of* S_i*. The decision problem is to decide whether we can identify a subset* $F \subseteq S$ *such that* $\cup_{S_i \in F} S_i = U$*, and the sum of the cost of* F*, i.e.,* $\sum_{S_i \in F} c_i$*, is minimized.*

Next, we reduce the WSC problem to an LCSK query q *by two steps. Firstly, we show how to build the query* $q = (\ell, \omega, W, \theta)$ *based on* U, S *and* C*. All elements in* U *correspond to the set of query keywords, i.e.,* ω*. We then set the weight vector* W *as* $\{1, 0, 0, 0, 0\}$ *and the threshold* $q.\theta$ *to 1. Here,* $q.\ell$ *can be assigned with any location, which has no influence. Secondly, the spatial database* O *can be constructed as follows. We observe that each set* S_i *corresponds to an object* o_i *and the set of keywords* $o_i.\omega$ *is comprised of all elements in* S_i*. We then assign* c_i *as the cost distance* $cd(o_i, q)$ *and set each level of* $o_i.\nu$ *to 1. It is clear that there is a solution to the WSC problem if and only if there is an answer to the query* q*. Hence, we complete the proof.*

4. Exact Algorithm

Motivated by the observation that we only need to consider RO_q for a specific query q, we develop an exact algorithm called MergeList for the case when $|RO_q|$ is limited. We first introduce the keyword hash table (KHT) index structure in Section 4.1, and then elaborate MergeList in Section 4.2.

4.1. The KHT Index Structure

To facilitate the access to the objects in RO_q, we propose the KHT index structure. It uses the hash table to organize the objects in O. Table 2 shows that each entry of KHT is of the form $< eid, t, olist, fi, si >$, where eid is the identifier of an entry and t denotes a keyword followed by a list of objects $olist$. $olist$ maintains these objects that contain t in their textual descriptions, i.e., $olist = \{o | o \in O \wedge t \in o.\omega\}$. fi and si indicate the first and second index subscripts of an entry, respectively. That is, we identify an entry of KHT by a two-level index.

To manage the trade-off between the exploration time and the space consumption, we use the perfect hashing technique [16] to build the two-level index. In particular, each entry e is projected to a two-level index (fi, si) based on the associated keyword t, and thus can be accessed in $O(1)$ time. As shown in Fig. 2, we map the keyword t to a two-level index in two steps. First, we project t to a unique integer $FTemp$ using the string hash function BKDRHash. Note that, other string hash functions can also be exploited. Next, we derive fi by performing a modulus on $FTemp$ over M, where M is a predefined cardinality for the first level index. That is, the range of the first index varies from 0 to $M - 1$. Second, we take $FTemp$ as the parameter and generate $STemp$ using an integer hash function. Similarly, we perform a modulus on $STemp$ over $C(fi)$ to get si. $C(fi)$ indicates the capacity of the first level index fi, which is dynamically

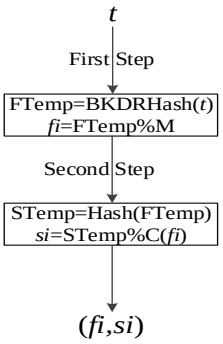


Fig. 2: Construction of a two-level index

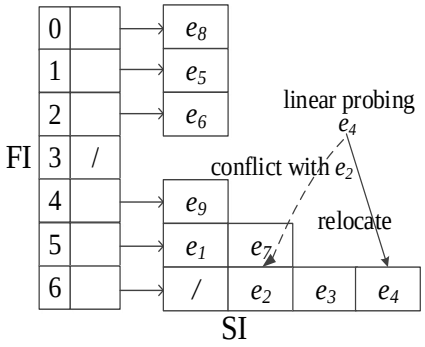


Fig. 3: An overview of KHT structure

determined based on the perfect hashing technique. For example, if there are k keywords are projected to fi after the first step, then $C(fi) = k^2$. By these two steps, each entry is associated with a two-level index, denoted by (fi, si).

As an example, Fig. 3 presents an instance of KHT index over the entries in Table 2. For reason of space, we denote by *eid* the corresponding entry and omit the empty entries in Fig. 3. It is noteworthy that a delicate situation arises when multiple entries share the same two-level index (fi, si), and the possibility is less than 0.5 as pointed out by [16]. We use the *linear probing* technique to tackle this problem. For instance, there is a conflict when we attempt to insert e_4 into KHT since e_2 has been already in the location $(6, 1)$. As the location $(6, 2)$ is occupied as well, with the linear probing technique, e_4 is inserted into $(6, 3)$, as shown by the solid arrow. With KHT, given a query q, we can retrieve RO_q in $O(|q.\omega|)$ time and avoid unnecessary accesses significantly.

4.2. Query Processing of MergeList

We first consider a naive exact algorithm that enumerates all subsets of RO_q, and then delivers the optimal one that satisfies the threshold constraint and has the smallest cost distance as the answer. Fig. 4 illustrates how to enumerate all subsets iteratively. Initially, the *candidate set* $\tilde{C}$ only has one element, i.e., the empty set $\{\}$. In each of the subsequent iterations, for each existing subset (e.g., $\{\}$) in $\tilde{C}$, it is combined with the current visited object (e.g., o_1) to generate a new subset (e.g., $\{o_1\}$). The procedure terminates once all the objects in RO_q have been visited, and the optimal one in $\tilde{C}$ is delivered as the answer.

Clearly, the naive method yields an exponential time complexity in terms of $|RO_q|$. The bottleneck lies in that all subsets of RO_q have to be enumerated and checked for finding the answer. In practice, however, a large number of the subsets are unqualified. They can be pruned by the best answer found so far if they have an equal or greater cost distance. Inspired by this, we develop an enhanced algorithm called MergeList, which prunes the candidate space with several strategies.

To answer the LCSK query, MergeList first places the objects in RO_q in ascending order of their cost distances, and then updates the candidate set $\tilde{C}$ iteratively by the objects in RO_q until the termination condition (to be presented in Lemma 2) is achieved. Note that we keep the best answer (a subset of RO_q) found so far and its cost distance in COS and $minCost$, and deliver COS as the answer.

MergeList enhances the naive approach from the following two aspects. First, MergeList organizes the objects in RO_q in ascending order of their cost distances. This enables two effective pruning strategies, namely Lemmas 1 and 2, to significantly shrink the candidate space. Second, $\tilde{C}$ only maintains the promising answers, rather than maintaining all the subsets of RO_q as in the naive approach. The *promising answer* ξ refers to a group of objects in RO_q satisfying two conditions: (1) the threshold constraint is not satisfied, that is, $\exists t \in q.\omega \wedge cov(\xi, t) < q.\theta$. As for ξ that satisfies the threshold constraint, we know that it is a feasible solution. Hence, we can update COS by it or prune it by COS, depending on their cost distances; (2) the cost distance of ξ is less than $minCost$, i.e., $\sum_{o \in \xi} cd(o, q) < minCost$. As for ξ that has an equal or greater cost distance, we can prune them by Lemma 1. Thus, they do not need to be reserved in $\tilde{C}$.

Step	Action	$\tilde{C}$
1	initialization	{}
2	visit o_1	{},{o_1}
3	visit o_2	{},{o_1},{o_2},{o_1,o_2}
.	.	.
.	.	.
.	.	.

Fig. 4: Illustration of the naive approach

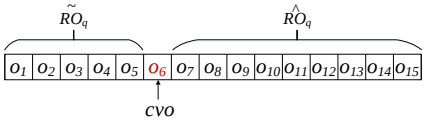


Fig. 5: Illustration of the RO_q

We proceed to introduce some notations used by the following lemmas. As shown in Fig. 5, the current visited object *cvo* splits the ordered RO_q into three components, namely $\tilde{RO}_q$, *cvo* and $\hat{RO}_q$. We denote by $\tilde{RO}_q$ the objects in RO_q that have been already visited (before *cvo*). Similarly, $\hat{RO}_q$ represents the objects that have not been visited (after *cvo*). In particular, given two objects o, $o' \in RO_q$, we denote by $o \prec o'$ ($o \preceq o'$) if o is to be visited after (not before) o' in RO_q.

We claim that a promising answer ξ *precedes cvo* if $\xi \subseteq \tilde{RO}_q$, denoted by $\xi|_{cvo}$. Put differently, $\forall o \in \xi$, it holds that $cov \prec o$. Consider, for example, the promising answer $\{o_1, o_5\}$ precedes o_6 in Fig. 5 because $o_6 \prec o_1$ as well as $o_6 \prec o_5$. Given a promising answer ξ (e.g., $\{o_1, o_5\}$) that precedes *cvo*, we say that the set ξ' (e.g., $\{o_1, o_5, o_8\}$) is a successor of ξ w.r.t. *cvo* (e.g., o_6), if $\xi \subset \xi'$ and $\forall o \in \xi' \wedge o \notin \xi$, it holds that $o \preceq cvo$. For instance, $\{o_1, o_5, o_8\}$ is a successor of $\{o_1, o_5\}$ w.r.t. o_6. Specifically, each promising answer will later spawn an exponential number of successors. We denote by Γ_ξ^{cvo} all the successors of ξ w.r.t. *cvo* hereafter.

Lemma 1. *Given a query q, a promising answer $\xi \in \tilde{C}$ and the current visited object $cvo \in RO_q$. If the sum of the cost distance of ξ and cvo is greater than minCost, that is, $\sum_{o\in\xi} cd(o,q)+cd(cvo,q) > minCost$, then no successor $\xi' \in \Gamma_\xi^{cvo}$ can be the answer of q.*

Proof. *We prove this lemma by contradiction. Let $\xi' \in \Gamma_\xi^{cvo}$ be the answer returned by MergeList. Thus, we have $\sum_{o\in\xi'} cd(o,q) \leq minCost$. By the property of successor, we know that there exists at least one object o', and it holds that $o' \in \xi' \wedge o' \notin \xi$. Also, we know that $o' \prec cov$, and thus $cd(cov,q) \leq cd(o',q)$. Combining the above two inequalities together with the given condition $\sum_{o\in\xi} cd(o,q)+cd(cvo,q) > minCost$, we have $\sum_{o\in\xi'} cd(o,q) = \sum_{o\in\xi} cd(o,q) + \sum_{o'\in\xi'\wedge o'\notin\xi} cd(o',q) > minCost$. This contradicts the assumption that ξ' is the final answer. Thus, no successor $\xi' \in \Gamma_\xi^{cvo}$ can be the answer. We complete the proof.*

Lemma 1 enables to significantly shrink the candidate space, namely the size of $\tilde{C}$. As suggested in Lemma 1, if the sum of the cost distance of ξ and *cvo* is greater than *minCost*, then we can discard ξ and all its successors Γ_ξ^{cvo} from consideration. This is because the solution quality of *COS* is always better than or equal to that of any set in $\xi \cup \Gamma_\xi^{cvo}$. Subsequently, we reveal an appealing property that allows to terminate the query processing.

Lemma 2. *Given a query q and the ordered RO_q, in which the objects are placed in ascending order of their cost distances. If $\forall \xi \in \tilde{C}$, the sum of the cost distance of ξ and cvo is greater than minCost, i.e., $\sum_{o\in\xi} cd(o,q) + cd(cvo,q) > minCost$, then COS is the answer.*

Proof. *As noted earlier, $\tilde{C}$ maintains all the promising answers found so far. Let ξ be any promising answer in $\tilde{C}$. By the property of the promising answer, we know that ξ is not a feasible solution, and additional objects that are not before cvo, i.e., $o' \preceq cvo$, are required to satisfy the threshold constraint. On the other hand, we know that $\forall o' \preceq cvo$, it holds that $cd(o',q) \geq cd(cvo,q)$. Based on the given condition $\sum_{o\in\xi} cd(o,q)+cd(cvo,q) > minCost$, we derive that $\sum_{o\in\xi} cd(o,q)+cd(o',q) > minCost$ as well. Combining the above inequality together with Lemma 1, we derive that there does not exist unseen feasible solutions that achieve a smaller cost distance than minCost. Thus, COS is the final answer and we complete the proof.*

Algorithm 1: *MergeList*

Input : KHT, q
Output: G

1 $COS \longleftarrow \{\}$; $minCost \longleftarrow \infty$;
2 add the empty set $\{\}$ into $\tilde{C}$;
3 $RO_q \longleftarrow$ retrieve the relevant objects to q from KHT;
4 place the objects in RO_q in ascending order of their cost distances;
5 **for** *each object* $cvo \in RO_q$ **do**
6 **if** $cd(cvo, q) > minCost$ **then** break;
7 **for** *each* $\xi \in \tilde{C}$ **do**
8 **if** $\sum_{o \in \xi} cd(o, q) + cd(cvo, q) > minCost$ **then**
9 delete ξ from $\tilde{C}$;
10 continue;
11 $tempSet \longleftarrow \xi \cup \{cvo\}$;
12 **if** *tempSet is a feasible solution* **then**
13 $COS \longleftarrow tempSet$;
14 $minCost \longleftarrow$ the cost distance of $tempSet$;
15 delete ξ from $\tilde{C}$;
16 **else** add $tempSet$ into $\tilde{C}$;
17 $G \longleftarrow COS$;
18 return G as the answer;

We elaborate the details of MergeList in Algorithm 1. We start by constructing the ordered RO_q (lines 3-4), in which the objects are placed in ascending order of their cost distances. We then access the objects in RO_q iteratively as follows. Consider an iteration. We first examine whether the cost distance of cvo is greater than $minCost$. If so, for $\forall \xi \in \tilde{C}$, we know that the sum of the cost distance of ξ and $cd(cvo, q)$ is greater than $minCost$ as well. Based on Lemma 2, we know that COS is the answer, thus we terminate the procedure (line 6). Otherwise, each $\xi \in \tilde{C}$ is processed as follows. We check whether Lemma 1 is satisfied. If yes, we delete ξ from $\tilde{C}$ (lines 8-10). If no, we combine cvo with ξ to generate a new set $tempSet$ (line 11). In particular, if $tempSet$ is a feasible solution, we update COS and $minCost$ accordingly by $tempSet$ (lines 12-15). Otherwise, we add $tempSet$ into $\tilde{C}$ for further exploration. Finally, we assign G as COS, and deliver G as the answer (lines 17-18).

Example 2: Consider a query q associated with two query keywords, e.g., $q.\omega = \{t_1, t_2\}$. Fig. 6(a) shows the objects in RO_q, the cost distance to q and the coverage weight of these two keywords by objects in RO_q. In Fig. 6(b), we present how to answer the query by MergeList.

- Step 1 (initialization): We initialize the Candidate Set $\tilde{C}$, $minCost$ and COS at this step.
- Step 2 (visit o_1): At this step, we generate a new set for each promising answer in $\tilde{C}$ by combining o_1 with it. For instance, we combine o_1 with the promising answer $\{\}$ and obtain $\{o_1\}$.
- Step 3 (visit o_2): Similarly, at this step, we combine o_2 with all existing promising answers in $\tilde{C}$ as shown in Fig. 6(b).
- Step 4 (visit o_3): At this step, we derive a feasible solution, i.e., $\{o_2, o_3\}$. We assign COS as $\{o_2, o_3\}$, and then prune the unqualified elements, e.g., $\{o_1, o_2, o_3\}$ by Lemma 1. Note that, we delete $\{o_2, o_3\}$ from $\tilde{C}$ as well. This is because each successor of $\{o_2, o_3\}$ has a greater cost distance than $\{o_2, o_3\}$, and cannot be the answer. Thus, we can stop the expansion from $\{o_2, o_3\}$ by deleting it from $\tilde{C}$.
- Step 5 (visit o_4): At this step, we prune unqualified elements of $\tilde{C}$ by Lemma 1. For instance, as the sum of the cost distance of $\{o_1, o_2\}$ and $\{o_4\}$ is greater than $minCost$, we delete $\{o_1, o_2\}$ from $\tilde{C}$.
- Step 6 (visit o_5): Since the cost distance of o_5 is greater than $minCost$, Lemma 2 is applied and we terminate the procedure. Finally, we deliver $\{o_2, o_3\}$ as the answer.

Object id	o_1	o_2	o_3	o_4	o_5
Cost distance to q	2	2.5	4	5	7
Coverage weight	0.1, 0.0	0.1, 0.3	0.3, 0.1	0, 0.3	0.1, 0.3

(a) Details of the objects in RO_q

Step	Action	Candidate Set	*minCost*	*COS*
1	initialization	{}	∞	{}
2	visit o_1	{},{o_1}	∞	{}
3	visit o_2	{},{o_1},{o_2},{o_1, o_2}	∞	{}
4	visit o_3	{},{o_1},{o_2},{o_1, o_2},{o_3},{o_1, o_3}	6.5	{o_2, o_3}
5	visit o_4	{},{o_4}	6.5	{o_2, o_3}
6	visit o_5	{},{o_4}	6.5	{o_2, o_3}

(b) Steps of the query processing

Fig. 6: Illustration of the MergeList algorithm

Theorem 2. *(Correctness of MergeList) MergeList always returns the correct answer.*

PROOF. *Let the cardinality of RO_q be n, thus there are up to $2^n - 1$ non-empty elements in $\tilde{C}$. For any $\xi \in \tilde{C}$, it is either used to update COS or pruned by COS. Specifically, if the sum of the cost distance of ξ and cvo is less than minCost, we update COS by $\xi \cup \{cvo\}$. Otherwise, we delete ξ from $\tilde{C}$ based on Lemma 1. Hence, it suffices to show that MergeList never prunes any feasible solution with a smaller cost distance than minCost (false negatives), and never maintains a feasible solution with a greater cost distance than minCost (false positives). Combining the above two observations, we safely drive that MergeList always returns the correct answer.*

5. Approximate Algorithm

MergeList produces and checks a large number of promising subsets for answering a query, which is prohibitively expensive when the dataset is large. We thus devise an approximate algorithm called MaxMargin. In the following, we introduce the proposed LIR-tree in Section 5.1, and elaborate optimizing strategies in Section 5.2. Finally, we delve into the details of MaxMargin in Section 5.3 with theoretical analysis.

5.1. The LIR-tree Index Structure

We proceed to briefly review the IR-tree [14], which lays the foundation of the LIR-tree. The IR-tree is essentially an R-tree [22] with each node augmented with an inverted file [57]. Each node n of the IR-tree contains multiple entries. Each entry e is of the form $(id, mbr, text)$, where id is an identifier of a child node (or an object) in the non-leaf (or leaf) node, mbr is the minimum bounding rectangle (MBR) enclosing all rectangles of the entries in n, and $text$ captures the textual description of all entries in n. In addition, each node is associated with an inverted file constituted by two components, namely the vocabulary table and the posting list, as follows:

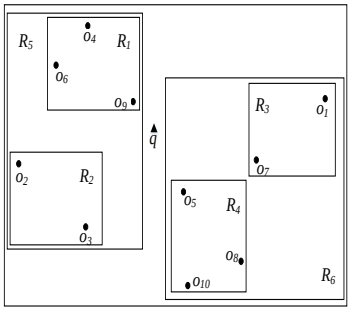
- Vocabulary table: Including all distinct keywords contained by the textual description of objects in the underlying database.
- Posting list: For each keyword t, the corresponding posting list maintains all the ids of the entries (i.e., the objects or the child nodes) with t in their descriptions.

The IR-tree enables to prune the search space from a high-level perspective. Consider, for example, we can first check whether a node n satisfies the query constraints, if so we add all its child nodes into the priority queue for further exploration. Otherwise, we skip the entire subtree rooted at n.

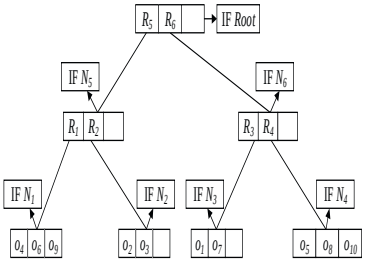
To be adaptive to our problem, we propose the LIR-tree that is extended from the IR-tree. The major difference between the IR-tree and LIR-tree comes from the inverted file. As discussed before, the posting list of inverted file only maintains the ids for each IR-tree node. This is not enough in our settings where the keyword level and the cost of objects are considered. We thus embed the level and cost into the posting lists. Note that, we maintain the information of level and cost in leaf nodes, and only maintain the cost in non-leaf nodes. This is because the level is meaningful for objects rather than nodes.

Object descriptions			
object	keywords	level vector	cost
o_1	t_1, t_3	3, 1	0.4
o_2	t_1, t_6	1, 1	0.2
o_3	t_2, t_5	5, 2	0.7
o_4	t_3, t_7	2, 3	0.5
o_5	t_4, t_7	2, 4	0.6
o_6	t_3, t_5	5, 3	0.8
o_7	t_2, t_6	1, 2	0.3
o_8	t_5, t_6	1, 5	0.6
o_9	t_1, t_7	3, 2	0.5
o_{10}	t_6, t_7	4, 3	0.7

(a) Object descriptions



(b) The MBR of objects



(c) The LIR-tree

Fig. 7: An instance of the LIR-tree

IF N_1	IF N_2	IF N_3	IF N_4	IF N_5	IF N_6	IF *Root*
t_1:0.5;(o_9,3)	t_1:0.2;(o_2,1)	t_1:0.4;(o_1,3)	t_4:0.6;(o_5,2),(o_{10},4)	t_1:0.2;(N_1,0.5),(N_2,0.2)	t_1:0.4;(N_3,0.4)	t_1:0.2;(N_5,0.2),(N_6,0.4)
t_3:0.5;(o_4,2),(o_6,5)	t_2:0.7;(o_3,5)	t_2:0.3;(o_7,1)	t_5:0.6;(o_8,1)	t_2:0.7;(N_2,0.7)	t_2:0.3;(N_3,0.3)	t_2:0.3;(N_5,0.7),(N_6,0.3)
t_5:0.8;(o_6,3)	t_5:0.7;(o_3,2)	t_3:0.4;(o_1,1)	t_6:0.6;(o_8,5)	t_3:0.5;(N_1,0.5)	t_3:0.4;(N_3,0.4)	t_3:0.4;(N_5,0.5),(N_6,0.4)
t_7:0.5;(o_4,3),(o_9,2)	t_6:0.2;(o_2,1)	t_6:0.3;(o_7,2)	t_7:0.6;(o_5,4),(o_{10},4)	t_5:0.7;(N_1,0.8),(N_2,0.7)	t_4:0.6;(N_4,0.6)	t_4:0.6;(N_6,0.6)
				t_6:0.2;(N_2,0.2)	t_5:0.6;(N_4,0.6)	t_5:0.6;(N_5,0.7),(N_6,0.6)
				t_7:0.5;(N_1,0.5)	t_6:0.3;(N_3,0.3),(N_4,0.6)	t_6:0.2;(N_5,0.2),(N_6,0.3)
					t_7:0.6;(N_4,0.6)	t_7:0.5;(N_5,0.5),(N_6,0.6)

Fig. 8: Content of augmented inverted files

In leaf nodes, the posting list corresponding to keyword t is of the form $< c_t; (o_{i_1}, l_{i_1}), ..., (o_{i_n}, l_{i_n}) >$, where c_t maintains the smallest cost of objects in the posting list (i.e., from o_{i_1} to o_{i_n}). The pair (o_{i_k}, l_{i_k}) records the id of an object and the corresponding level of t in o_{i_k}. We consider IF N_4 in Fig. 8 as an example. We observe that the posting list of t_4 contains two objects, namely o_5 and o_{10}. The c_{t_4} is set to 0.6, which is the smallest cost of objects o_5 and o_{10} (see Fig. 7(a)). Also, we know that o_5 is associated with two keywords t_4 and t_7, and the level of t_4 is 2. We thus maintain the pair $(o_5, 2)$ in the posting list of t_4.

In non-leaf nodes, the posting list corresponding to keyword t is of the form $< \tilde{c_t}; (n_{i_1}, c_{i_1}), ..., (n_{i_n}, c_{i_n}) >$, where $\tilde{c_t} = \min_{1<k<n} c_{i_k}$ is the smallest cost of all child nodes in the posting list. The pair (n_{i_k}, c_{i_k}) records the id of a child node and the corresponding smallest cost w.r.t. the keyword t (i.e., c_t) in the inverted file of n_{i_k}. The IF N_6 in Fig. 8 shows that the posting list of t_6 contains two entries, namely $(N_3, 0.3)$ and $(N_4, 0.6)$. That is to say that the cost of N_3 is 0.3 w.r.t. t_6, which can be obtained from the posting list of t_6 in IF N_3. Also, we set $\tilde{c_{t_6}}$ to 0.3, since it holds that $0.3 < 0.6$.

5.2. Optimizing Strategies

Literatures [6, 7] devise greedy algorithms for answering the CoSKQ, which iteratively append the current optimal entry into the result set by traversing the IR-tree in the best-first fashion. Initially, the root node of IR-tree is pushed into the priority queue Q. In each of the subsequent iterations, the top entry e is popped and examined as follows: If e is an object, we add it into the result set G and update all the remaining entries in Q accordingly. Otherwise, all the child nodes of e are pushed into Q for further exploration.

Two major drawbacks degrade the performance of the above algorithms: (1) the pruning power of the algorithms is insufficient, and (2) they have to update all the remaining entries in each iteration, which is computationally expensive. We develop two strategies, namely BBS and TUS, to attack the aforementioned drawbacks accordingly.

Definition 4. (Minimum Cost Distance) Given a query q and a node n of the LIR-tree, we define the minimum cost distance between n and q as:

$$mcd_q(n) = minDist(q, n) * minCost(q, n) \tag{3}$$

In Equation (3), we denote by $minDist(q, n)$ the minimum distance between q and the MBR of n. If q falls into the MBR of n, we set this distance to 0. Otherwise, the distance can be set as the minimum distance between q and the edges of the MBR of n. The $minCost(q, n)$ refers to the smallest cost of objects in n that are relevant to the query q. Recall that, the smallest cost is maintained in the posting list (see Fig. 8), and thus we can get $minCost(q, n)$ easily. Based on the above discussion, we know that $mcd_q(n)$ is the lower bound of the cost distance of all objects in n. For simplicity, *we say that an object o is in n if o is enclosed by the MBR of n hereafter*, whenever there is no ambiguity.

Lemma 3. *Let FS be a feasible solution to a query q, $\hat{FS}$ be the corresponding cost distance, and n be a node of the LIR-tree. If $mcd_q(n) > \hat{FS}$, then no object in n can be in the answer.*

PROOF. *Recall that, $mcd_q(n)$ offers a lower bound of the cost distance of all objects in n. That is, for any object o in n, it holds that $cd(o, q) \geq mcd_q(n)$. Combining the given condition $mcd_q(n) > \hat{FS}$, we derive $cd(o, q) > \hat{FS}$ as well. As FS is a feasible solution to q, and it achieves better performance than any other feasible solutions that contain the objects in n in terms of the cost distance. As a result, no object in n can be in the answer and thus can be discarded from consideration. Hence, we complete the proof.*

BBS: As suggested in Lemma 3, we can take $\hat{FS}$ as the upper bound and prune the search space with it. In particular, we examine each node n as follows: If $mcd_q(n) > \hat{FS}$, we then skip the entire subtree rooted at n and all objects in n are pruned. Otherwise, we add all the child nodes of n into the priority queue Q for further exploration. Clearly, BBS facilitates the query processing by reducing the cardinality of the priority queue Q.

Next, we describe how to build the initial feasible solution. We build it with the relevant objects that are close to q. We start from the root node of the LIR-tree, and then find the unseen *nearest leaf node* iteratively in the best-first fashion. We denote by the *nearest leaf node* a leaf node n that has the smallest $minDist(q, n)$ and contains at least one query keyword. Then, we place all relevant objects in n in ascending order of their cost distances to q, and add these objects into FS progressively until the threshold constraint is satisfied. That is, FS has been already a feasible solution. Note that, we continue to find the next unseen nearest leaf node if the threshold constraint is not satisfied, and repeat the above procedure.

Before describing in detail how to use TUS to further improve the query performance, we first introduce some concepts.

Definition 5. (Contribution Ratio) Given a query q and an entry e (i.e., an object or a node) of the LIR-tree, we define the contribution ratio cr of e to q as follows:

$$cr(e, q) = \begin{cases} \frac{|q.\omega| * q.\theta}{mcd_q(e)} & \text{if e is a node,} \\ \frac{cov(e,q)}{cd(e,q)} & \text{if e is an object.} \end{cases}$$

In Definition 5, we denote by $|q.\omega|$ the number of query keywords. By considering both the coverage weight and the cost distance, $cr(e, q)$ can better capture the contribution of e to q than that of $cov(e, q)$. Note that the contribution ratio of an unseen entry e in Q decreases as the objects are added into G. To capture this change dynamically, we define the *dynamic contribution ratio*.

Definition 6. (Dynamic Contribution Ratio) Given a query q and an entry e (i.e., an object or a node) of the LIR-tree, we define the dynamic contribution ratio dcr of e when there are r objects in the result set G, i.e., $|G| = r$, as follows:

$$dcr_q^r(e) = \begin{cases} \frac{\Upsilon_q^r * q.\theta}{mcd_q(e)} & \text{if e is a node,} \\ \frac{cov^r(e,q)}{cd(e,q)} & \text{if e is an object.} \end{cases}$$

In Definition 6, we denote by Υ_q^r the number of query keywords whose coverage weights have not reached the threshold $q.\theta$. The $cov^r(e, q)$ represents the remaining coverage weight of q by the object e when r objects are included in G. Note that, Υ_q^r and $cov^r(e, q)$ keep decreasing when we add an object into G.

Algorithm 2: *MaxMargin*

Input : q
Output: G

```
1  r ⟵ 0; G ⟵ ∅;
2  for i ⟵ 1 to |q.ω| do RV[i] ⟵ q.θ;
3  construct the feasible solution FS with the objects near to q;
4  upBound ⟵ Σ_{o∈FS} cd(o, q);
5  Q.enqueue(LIR.root);
6  while (!Q.empty()) do
7      e ⟵ Q.dequeue();
8      if e is a node then
9          recompute the dcr_q^r(e);
10         if dcr_q^r(e) is greater than the dcr of the top entry in Q then
11             for each entry e' in node e do
12                 if ∃t ∈ q.ω ∧ t ∈ e'.text and mcd_q(e') < upBound then
13                     compute the dcr_q^r(e');
14                     Q.enqueue(e');
15         else Q.enqueue(e);
16     else
17         if e.cv[i] ≤ RV[i] for all i then
18             G ⟵ G ∪ {e};
19             for j ⟵ 1 to |q.ω| do
20                 if RV[j] ≥ e.cv[j] then RV[j] ⟵ RV[j] − e.cv[j];
21                 else RV[j] ⟵ 0;
22             if RV[j] == 0 for all j then break;
23             r++;
24             FS ⟵ refineFS(FS, G);
25             upBound ⟵ Σ_{o∈FS} cd(o, q);
26         else
27             for j ⟵ 1 to |q.ω| do
28                 if RV[j] < e.cv[j] then e.cv[j] ⟵ RV[j];
29             recompute the dcr_q^r(e);
30             Q.enqueue(e);
31 return G as the answer;
```

Lemma 4. *Given a query q, a node n of the LIR-tree. For any object o in n, and any r ($r \geq 0$), it holds that $dcr_q^r(o) \leq dcr_q^r(n)$.*

PROOF. *For any object o in n, we know that $cov^r(o,q) \leq \Upsilon_q^r * q.\theta$. Then, according to the definition of $mcd_q(n)$, we know that $mcd_q(n)$ is the lower bound of the cost distance of all objects in n, that is, $cd(o,q) \geq mcd_q(n)$. With these two inequalities, we have $dcr_q^r(o) \leq dcr_q^r(n)$. Hence, the proof finishes.*

Lemma 4 provides an upper bound for the dynamic contribution ratio of all the objects in n. Subsequently, we reveal an appealing property of the dynamic contribution ratio in Lemma 5. Based on these two lemmas, we develop TUS that reduces the updating cost of the priority queue Q.

Lemma 5. *Given a query q, an entry e (i.e., an object or a node) of the LIR-tree and two integers m, n ($m \leq n$), it holds that $dcr_q^n(e) \leq dcr_q^m(e)$.*

PROOF. *As suggested by the formula of dynamic contribution ratio, the denominator (e.g., cd(e,q)) is a constant. However, the numerator (e.g., $cov^r(e,q)$) may decrease when an object is added into G. As a*

Algorithm 3: $refineFS$

Input : FS, G
Output: FS'

1 $FS' \longleftarrow FS \cup G$;
2 organize the objects in FS' in descending order of their cost distances;
3 **for** *each object* $o \in FS'$ **do**
4 $\quad Temp \longleftarrow FS' - \{o\}$;
5 $\quad$ **if** $Temp$ *is a feasible solution* **then** $FS' \longleftarrow Temp$;
6 **if** $\sum_{o \in FS} cd(o, q) < \sum_{o \in FS'} cd(o, q)$ **then** $FS' \longleftarrow FS$;
7 return FS';

result, we have $cov^n(e, q) \leq cov^m(e, q)$ *and* $\Upsilon_q^n \leq \Upsilon_q^m$*, which leads to the decrease of dynamic contribution ratio. When* $m \leq n$*, we can safely draw the conclusion that* $dcr_q^n(e)$ *is not greater than* $dcr_q^m(e)$*. Hence, we complete the proof.*

TUS: Previous approaches in [6, 7] update all the remaining entries in Q when an object is added into G, which is computationally expensive. TUS does not update the dynamic contribution ratio of e until it is popped from the priority queue Q. Lemma 5 suggests that if e is still the current optimal entry among all the remaining entries in Q, we can handle it as follows: If e is an object, we then add it into G. Otherwise, we expand it by pushing all its child nodes into Q for further exploration. Specifically, the remaining entries in Q do not need to be updated immediately, which significantly reduces the updating overhead. We know that, TUS enhances the performance by reducing the updating cost of the priority queue.

5.3. Query Processing of MaxMargin

Based on the above two optimizing strategies, we develop the approximate algorithm MaxMargin. It answers the query by adding the current optimal object into the result set G progressively until the threshold constraint is satisfied.

Algorithm 2 presents the pseudocode for MaxMargin. We keep the relevant entries (i.e., the objects and nodes) in Q in descending order of the dynamic contribution ratio, and use RV to record the difference between $q.\theta$ and the coverage weight of each query keyword by G dynamically. Initially, each dimension of RV, e.g., $RV[i]$, is set to $q.\theta$ (line 2). Then, we build the initial feasible solution FS (as described in Section 5.2), and take *upBound* as the upper bound. Next, we perform an iterative procedure. We first pop the top entry e and find out whether e is a node. If yes, we continue as follows: We first recompute the dynamic contribution ratio of e, i.e., $dcr_q^r(e)$. If $dcr_q^r(e)$ is greater than the dynamic contribution ratio of the top entry in Q, that is, e is the current optimal entry. We thus push all unfiltered entries of e into Q (lines 10-14). Otherwise, e is pushed into Q (line 15). If no, that is, e is an object, there are two different cases. Case 1: If for $\forall i$, it holds that $e.cv[i] \leq RV[i]$. Here, we denote by $e.cv[i]$ the remaining coverage weight of the ith keyword in $q.\omega$ by e. We know that the dynamic contribution ratio of e has not decreased, and thus e does not need to be updated. We thus add e into G and update RV accordingly (lines 18-21). If the threshold constraint is achieved (line 22), we terminate the procedure immediately. Otherwise, we invoke Algorithm 3 to update FS and *upBound* (lines 24-25). Case 2: If $\exists i$, and it holds that $e.cv[i] > RV[i]$, we update RV and push the updated e into Q (lines 27-30).

Algorithm 3 shows how to update FS by G. We consider a simple strategy. First, FS is combined with G to form FS' (line 1). All objects in FS' are organized in descending order of their cost distances (line 2). Then, we remove the current visited object o from FS' and obtain a temporary set, namely $Temp$. We check whether $Temp$ is a feasible solution. If so, we then set FS' as $Temp$ (line 5). The algorithm repeats the above procedure until all objects in FS' have been explored. In line 6, if FS has a smaller cost distance, we update FS' by FS, which guarantees that we can always return a feasible solution whose cost distance is not larger than that of FS. Finally, we return FS' as a new feasible solution.

Theorem 3. *The approximation ratio of MaxMargin is not greater than* $\frac{H(\lfloor cov+1 \rfloor)}{q.\theta}$*, where cov is the largest* $cov(o_j, q)$ *for* $\forall o_j \in RO_q$*,* $cov + 1$ *is rounded down by* $\lfloor cov + 1 \rfloor$*, and* $H(k) = \sum_{i=1}^{k} \frac{1}{i}$ *is the kth harmonic number.*

PROOF. *Inspired by [13], we present the proof of this theorem as follows. We denote by m, n the number of elements in* $q.\omega$ *and* RO_q*, respectively. We define the* $m \times n$ *matrix* $P = (p_{ij})$ *as follows:*

$$p_{ij} = \begin{cases} cw(o_j, q.\omega^i) & \text{if } q.\omega^i \in o_j.\omega, \\ 0 & \text{otherwise.} \end{cases}$$

Based on the definition of P, we know that n columns of P correspond to n coverage weight vectors. MaxMargin is to find a group G of objects, and we use an incidence vector $x = (x_j)$ *to represent feasible solutions. As a result, the incidence vector x of any feasible solution satisfies:*

$$\begin{aligned} \sum_{j=1}^{n} p_{ij} x_j \geq q.\theta \quad & for\ all\ i, \\ x_j \in \{0, 1\} \quad & for\ all\ j. \end{aligned}$$

The formula above suggests that the group of objects x implied can satisfy the threshold constraint. For ease of presentation, we denote the cost distance of o_j *by* c_j *hereafter. Also, we denote by* cov_j^r *the remaining coverage weight to q by* o_j *when* $r - 1$ *objects are added into G. We claim that these inequalities imply*

$$\sum_{j=1}^{n} H(\lfloor cov_j^1 + 1 \rfloor) c_j x_j \geq q.\theta \sum_{o_j \in G} c_j \tag{4}$$

for the result set G returned by the greedy approach. Once (4) is proved, the theorem will follow by considering x be the incidence vector of an optimal feasible solution.

To prove (4), it is sufficient to exhibit nonnegative numbers $y_1, y_2, ..., y_m$ *such that*

$$\sum_{i=1}^{m} p_{ij} y_i \leq H(\sum_{i=1}^{m} p_{ij}) c_j \quad for\ all\ j \tag{5}$$

and such that

$$\sum_{i=1}^{m} y_i = \sum_{o_j \in G} c_j \tag{6}$$

for then

$$\begin{aligned} \sum_{j=1}^{n} H(\sum_{i=1}^{m} p_{ij}) c_j x_j &\geq \sum_{j=1}^{n} (\sum_{i=1}^{m} p_{ij} y_i) x_j \\ &= \sum_{i=1}^{m} (\sum_{j=1}^{n} p_{ij} x_j) y_i \\ &\geq q.\theta \sum_{i=1}^{m} y_i \\ &= q.\theta \sum_{o_j \in G} c_j \end{aligned}$$

as desired.

The numbers $y_1, y_2, ..., y_m$ satisfying (5) and (6) have a simple intuitive interpretation: each y_i can be interpreted as the cost distance achieved by MaxMargin for covering the keyword $q.\omega^i$. Without loss of generality, we can assume that G is $\{o_1,o_2,...,o_r\}$ when r objects are added into G, and so

$$\frac{cov_r^r}{c_r} \geq \frac{cov_j^r}{c_j}$$

for any r, j. If t objects are required to cover all query keywords, then

$$\sum_{o_j \in G} c_j = \sum_{j=1}^{t} c_j,$$

and

$$y_i = \sum_{r=1}^{t} \frac{c_r \cdot cw(o_r, q.\omega^i)}{cov_r^r}.$$

We know that

$$\sum_{i=1}^{m} y_i = \sum_{i=1}^{m} \sum_{r=1}^{t} \frac{c_r \cdot cw(o_r, q.\omega^i)}{cov_r^r} = \sum_{r=1}^{t} c_r$$

For any o_j, we know that cov_j^r decreases as the iteration continues from Lemma 5. We assume s is the largest superscript such that $cov_j^s > 0$ then

$$\begin{aligned}
\sum_{i=1}^{m} p_{ij} y_i &= \sum_{r=1}^{s} (cov_j^r - cov_j^{r+1}) \cdot \frac{c_r}{cov_r^r} \\
&\leq c_j \sum_{r=1}^{s} \frac{cov_j^r - cov_j^{r+1}}{cov_j^r} \\
&= c_j \sum_{r=1}^{s} \frac{cov_j^r - cov_j^{r+1}}{cov_j^r} \\
&\leq c_j \sum_{r=1}^{s} \frac{\lfloor cov_j^r + 1 \rfloor - \lfloor cov_j^{r+1} \rfloor}{\lfloor cov_j^r + 1 \rfloor} \\
&= c_j \sum_{r=1}^{s} \sum_{l=\lfloor cov_j^{r+1}+1 \rfloor}^{\lfloor cov_j^r+1 \rfloor} \frac{1}{cov_j^r} \\
&\leq c_j \sum_{r=1}^{s} \sum_{l=\lfloor cov_j^{r+1}+1 \rfloor}^{\lfloor cov_j^r+1 \rfloor} \frac{1}{l} \\
&= c_j H(\lfloor cov_j^1 + 1 \rfloor) - c_j H(\lfloor cov_j^s + 1 \rfloor) \\
&\leq c_j H(\lfloor cov_j^1 + 1 \rfloor)
\end{aligned}$$

Time Complexity. The time complexity of MaxMargin consists of two components, namely the cost of building the initial feasible solution and the cost of searching the answer in Q. Suppose that the height of the LIR-tree is l. Thus, there are $n = \frac{|O|}{2^{l-1}}$ objects in each leaf node on average. As discussed earlier, the relevant objects in a nearest leaf node need to be organized in ascending order of their cost distances, and then added into FS. In the worst case, the time complexity of building the feasible solution is $O(nlogn)$. As with [6], we assume that there are m relevant objects to q. That is, the number of entries (i.e., the objects

Table 3: Properties of real datasets

Property	BT	GN
# of objects	298,346	977,302
# of keywords	748,162	5,286,498
# of distinct keywords	58,367	116,466

Table 4: Parameter settings

Parameter	Range
DS (10^5)	0.1, **1**, 3, 5, 7, 9
TK	50, 100, 150, 200, 250, **300**
QK	2, **3**, 4, 5, 6, 7
TS	0.1, 0.2, **0.3**, 0.4, 0.5, 0.6
KD	3, **4**, 5, 6, 7, 8

and nodes) in the priority queue Q is $O(m)$. Given that the entry e may be reinserted into Q in MaxMargin, in the worst case each entry e will be reinserted into Q at most $O(m)$ times. Thus, there are at most $O(m^2)$ iterations before the threshold constraint is satisfied. In each iteration, the major overhead comes from reinserting e into the ordered Q, which costs $O(logm)$. Therefore, the time complexity of searching the answer is $O(m^2logm)$. As a result, the worst-case time complexity of MaxMargin is $O(nlogn + m^2logm)$.

6. Empirical Study

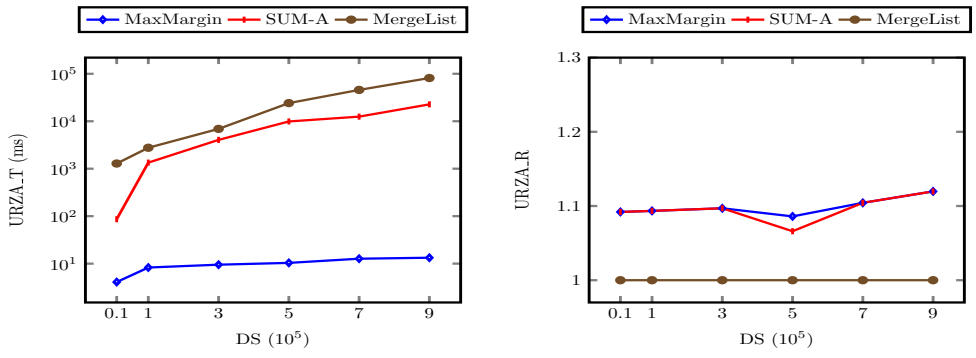
We conduct experiments over synthetic and real datasets to evaluate the performance of our proposed algorithms. We specify the experimental setup in Section 6.1, and report the results in Sections 6.2 and 6.3.

6.1. Experimental Setup

Algorithms. In the study of the LCSK query processing, we implement the state-of-the-art approximate algorithm SUM-A [6, 7], and compare it with MaxMargin. To be fair, we consider the modified version of SUM-A, which is based on the LIR-tree and can better match our settings. In addition, we consider MergeList as a comparison for evaluating the effectiveness of approximate algorithms.

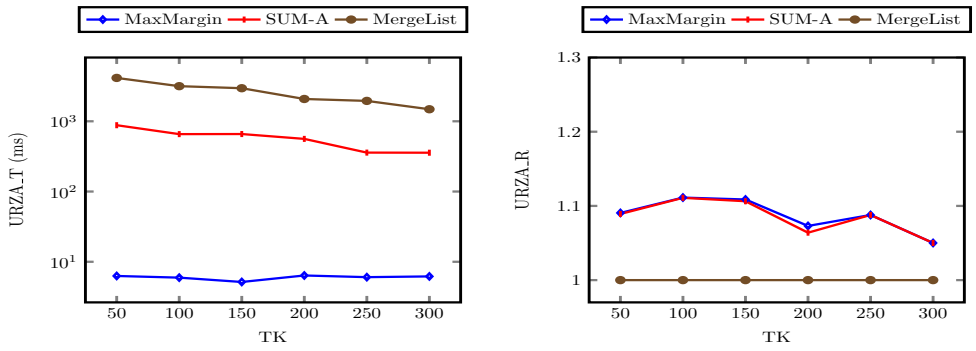
All algorithms were implemented in C/C++ and run in Windows 7 System on an Intel(R) Core(TM) i5-4590 CPU@3.30 GHz with 8GB RAM. Both KHT and LIR-tree are disk-resident, and the page size is set to 4 MB by default.

Datasets and queries. We deploy five synthetic and two real datasets in the experiments. Specifically, each synthetic dataset consists of three types of datasets following the uniform, random and zipf distributions. Table 3 presents the properties of real datasets, namely BT [32] and GN [6]. BT is extracted from OpenStreetMap and all points fall into the rectangle [(-11.1, 49.6),(2.1,62.5)]. Each pair of values (e.g., (-11.1,49.6)) corresponds to a coordinate of the form (*latitude, longitude*). The first coordinate corresponds to the bottom-left corner of the rectangle, and the second corresponds to the up-right corner. Each object



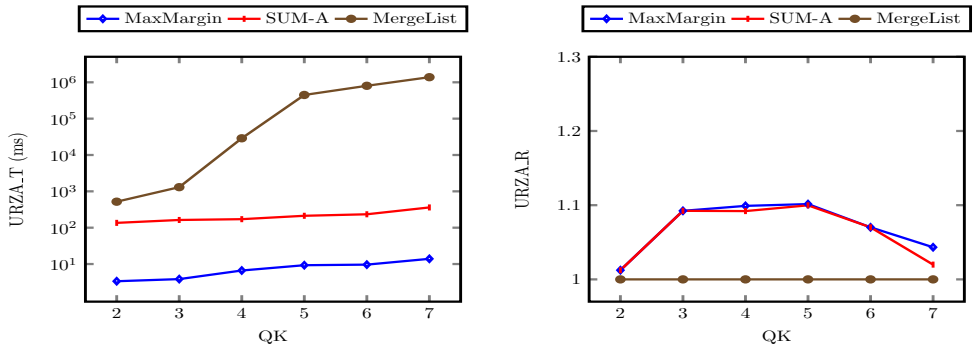
(a) Running time (b) Appro. ratio

Fig. 9: Effect of DS on synthetic datasets



(a) Running time (b) Appro. ratio

Fig. 10: Effect of TK on synthetic datasets



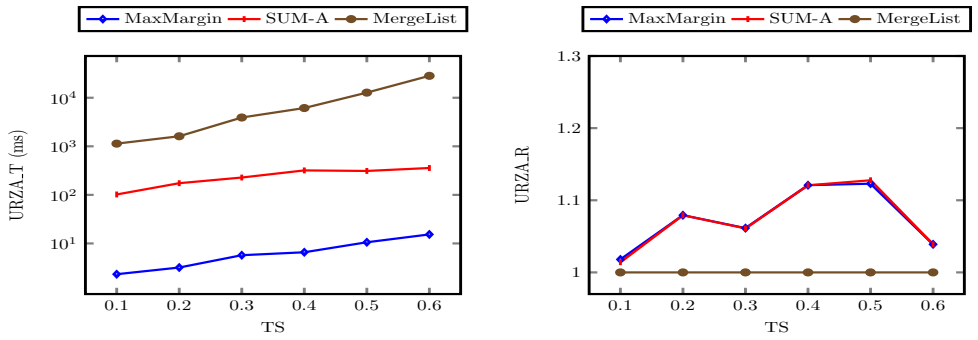
(a) Running time (b) Appro. ratio

Fig. 11: Effect of QK on synthetic datasets

in BT consists of a location and a set of keywords. GN is extracted from the U.S. Board on Geographic Names (geonames.usgs.gov). Similarly, each object is associated with a location and a textual description. We randomly generate the *cost* and *level vector* for the objects in the underlying datasets. Without loss of generality, we assume that the cost is in the range of $(0, 1)$. In addition, we assume the level ranges from 1 to 5. For example, all hotels can be classified into five categories based on the level of their services. Actually, this is not as restrictive as it seems, and the range can be extended to any scope.

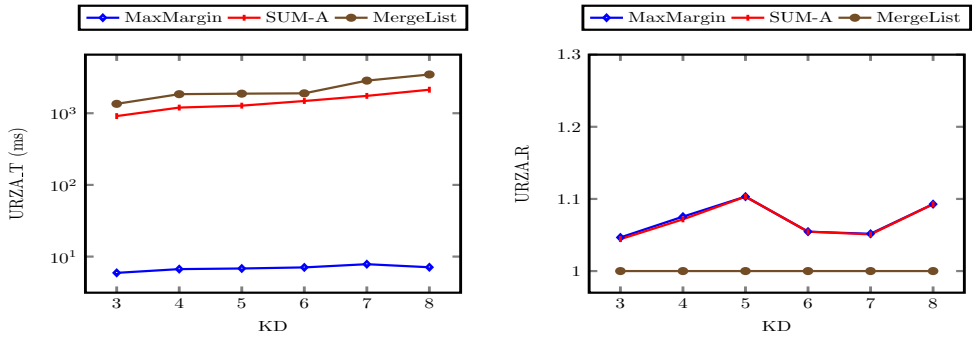
As illustrated in Table 4, we focus on measuring the impact of five parameters: (1) *data size* (DS); (2) *the total number of distinct keywords in the spatial database* (TK); (3) *the number of keywords associated with each object* (KD); (4) *the number of query keywords* (QK); (5) *the threshold of q, i.e., q.θ* (TS). All the default values are marked in bold in Table 4. We study all these parameters on synthetic datasets, but only study QK and TS on real datasets because other parameters (e.g., TK) are fixed in real datasets.

For each workload, we randomly generate 20 queries based on the parameter settings in Table 4. In synthetic datasets, we generate the query location ($q.\ell$) and keywords ($q.\omega$) randomly for each query q. In real datasets, however, a large number of the keywords are *rare*. That is, few objects contain them in their textual descriptions. Furthermore, these keywords are rarely considered as the query keywords by users. Motivated by these observations, we discard the keywords associated with at most 50 objects in the datasets from consideration. Then, we generate the query keywords randomly with the remaining keywords for each query q. The performance metrics that we measure are the response time and approximation ratio, and the results are computed by averaging the performance of 20 queries.



(a) Running time (b) Appro. ratio

Fig. 12: Effect of TS on synthetic datasets



(a) Running time (b) Appro. ratio

Fig. 13: Effect of KD on synthetic datasets

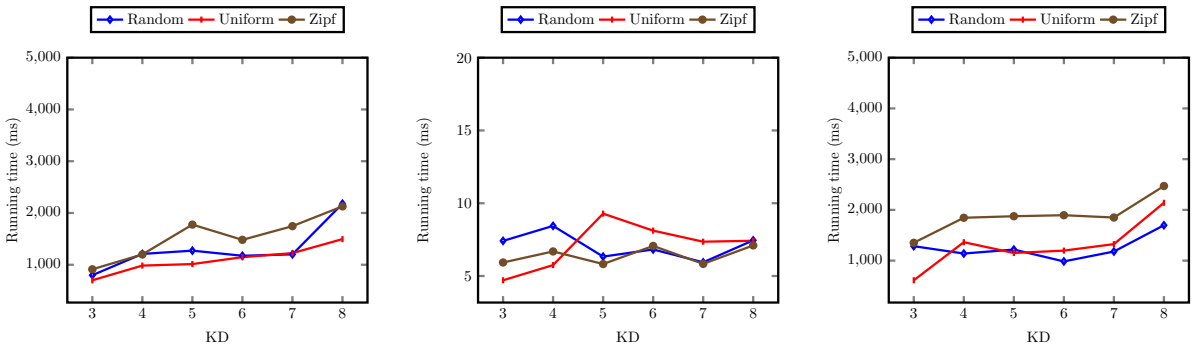
6.2. Results on Synthetic Datasets

We evaluate the overall performance of algorithms with three types of datasets, and report the average response time ($URZA\_T$) and average approximation ratio ($URZA\_R$), where $URZA\_T = \frac{T_u+T_r+T_z}{3}$, $URZA\_R = \frac{R_u+R_r+R_z}{3}$. We denote by T_u, T_r, T_z and R_u, R_r, R_z the response time and approximation ratio for uniform, random and zipf datasets, respectively.

Effect of DS. Fig. 9(a) shows that the response time of all algorithms increases as DS becomes larger. This is because that much more relevant objects are required to be explored. Specifically, MaxMargin runs several orders of magnitude faster than SUM-A, which is largely due to the two optimizing strategies used by MaxMargin. We present the approximation ratio of algorithms in Fig. 9(b), and denote by the curve $y = 1$ the approximation ratio of MergeList. Though both SUM-A and MaxMargin are based on the greedy strategy in [13], SUM-A achieves a little bit better performance than MaxMargin in terms of the accuracy. This might be due to the different strategies used to select the current optimal entry.

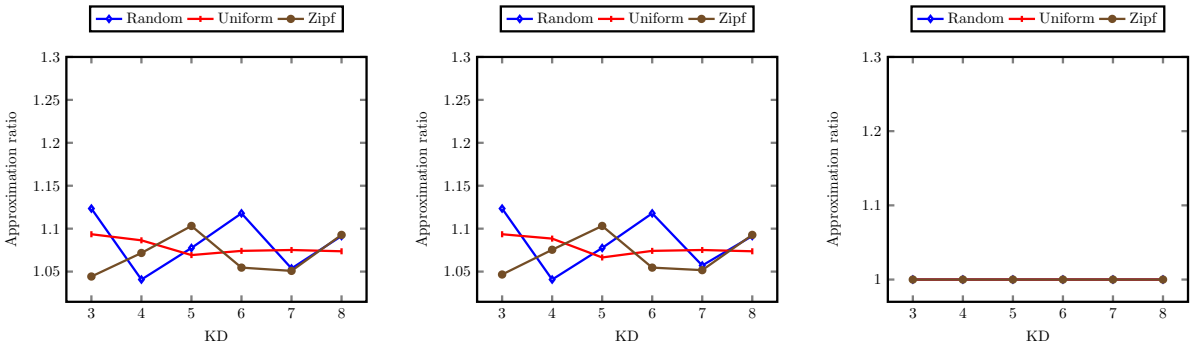
Effect of TK. When TK varies from 50 to 300, the experimental results are shown in Fig. 10. When TK increases, the response time of all algorithms decreases, as shown in Fig. 10(a). The reason behind is that the number of relevant objects, namely $|RO_q|$, becomes smaller when TK becomes larger. As for the accuracy, we notice that SUM-A and MaxMargin have the same performance in most cases in Fig. 10(b), and both of them achieve good accuracy bounded by the upper bound 1.2.

Effect of QK. The experimental results of algorithms when varying QK are shown in Fig. 11. When QK increases, the response time of the two approximate algorithms increases slightly, whereas the performance of MergeList degrades especially when QK varies from 3 to 5. Note that, we assign the default value of



(a) SUM-A (b) MaxMargin (c) MergeList

Fig. 14: Running time on different datasets



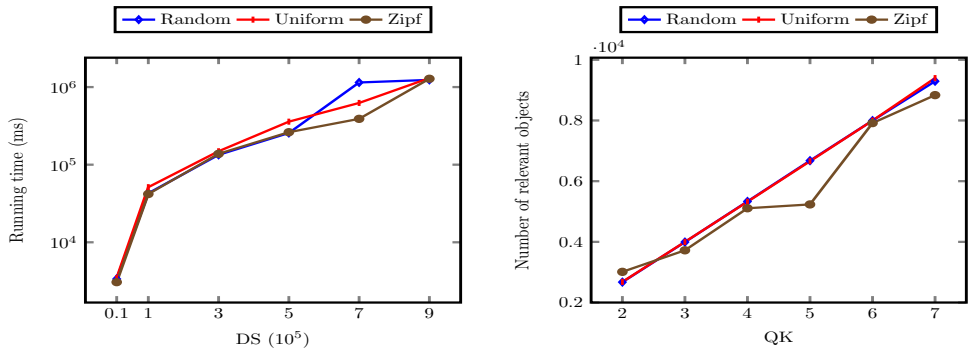
(a) SUM-A (b) MaxMargin (c) MergeList

Fig. 15: Approximation ratio on different datasets

KD as 4 in Table 4. Thus, when QK is less than 4, all query keywords may be covered by a single object. In contrast, when QK is greater than 4, much more objects are required to be explored for satisfying the threshold constraint. Consistently, we observe that the approximate algorithms achieve bad accuracy when QK varies from 3 to 5 (see Fig. 11(b)). Also, we know that the first selected optimal entry by the greedy strategy affects the subsequent selections, and thus has much effect on the accuracy. When much more objects are explored (when QK varies from 3 to 5), the first selected entry may not be optimal from the global view, which affects the whole performance of accuracy. Actually, this is an inherent limitation of the greedy strategy.

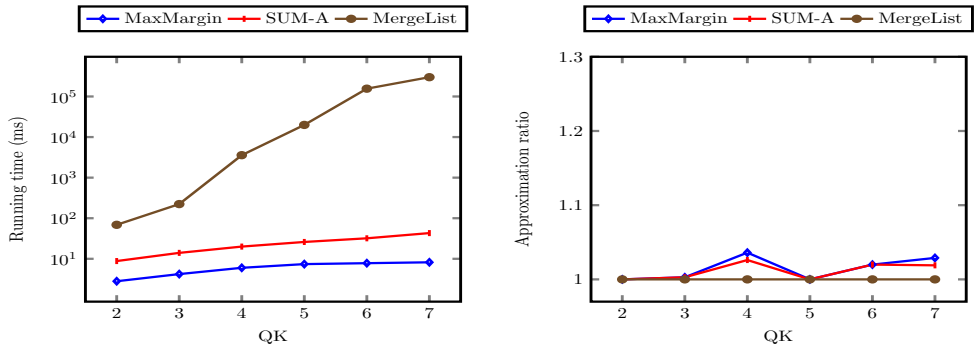
Effect of TS. We measure the impact of TS in this experiment, and present the results in Fig. 12. As expected, the response time of all algorithms increases as TS increases. This is because much more objects are explored before the threshold constraint is satisfied. Specifically, the response time of approximate algorithms increases almost linearly with TS, whereas MergeList increases rapidly. We observe that the accuracy of approximate algorithms fluctuates in Fig. 12(b), denoting that TS has much impact on the accuracy of algorithms. This is because it determines the cardinality of the result set G.

Effect of KD. As shown in Fig. 13(a), the response time of all algorithms increases almost linearly with KD. The reason behind is that the response time is proportional to the number of relevant objects, i.e., $|RO_q|$, which is often proportional to KD. One might wonder why does not MergeList increase rapidly in this experiment. This is largely due to the two pruning strategies used by MergeList. We also notice that MaxMargin runs much faster than SUM-A, which demonstrates the effectiveness of our proposed two optimizing strategies, namely BBS and TUS. The results in Fig. 13(b) show that both two approximate



(a) LIR-tree construction time (b) Number of relevant objects

Fig. 16: The performance on synthetic datasets



(a) Running time (b) Appro. ratio

Fig. 17: Effect of QK on BT

algorithms produce the near-optimal answers.

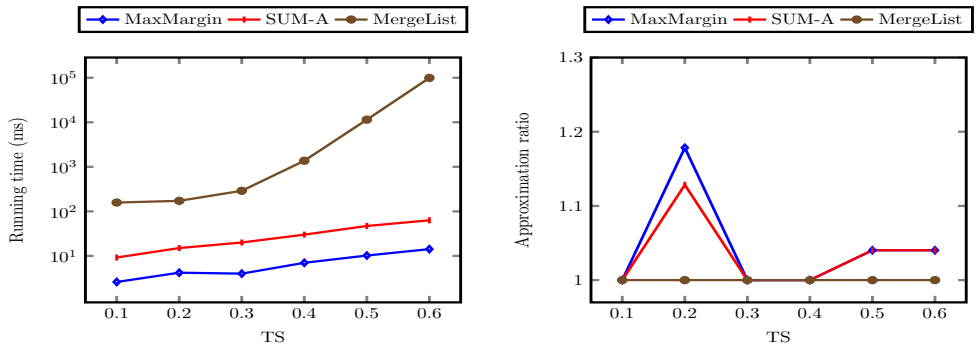
In the following, we delve more into details of the performance of algorithms on different types of datasets. Figs. 14 and 15 present the response time and accuracy of algorithms w.r.t. *KD*. Specifically, we study the performance of algorithms on different types of datasets, namely uniform, random and zipf. It is shown that all algorithms are sensitive to zipf datasets. This is because the distribution of zipf datasets has much effect on the pruning power of algorithms. We observe that the uniform datasets achieve more stable accuracy. This is largely due to the distribution of the datasets. Fig. 16(a) shows the LIR-tree index construction time by varing *DS*. In general, the construction time increases almost linearly with *DS* for all three types of datasets. In addition, Fig. 16(b) shows the number of relevant objects when increasing *QK*. Similarly, the number of relevant objects is proportional to the number of query keywords.

6.3. Results on Real Datasets

We proceed to study the performance of algorithms on BT and GN. As noted earlier, other parameters (e.g., *TK*) are fixed in real datasets, thus we only evaluate the effect of *QK* and *TS*.

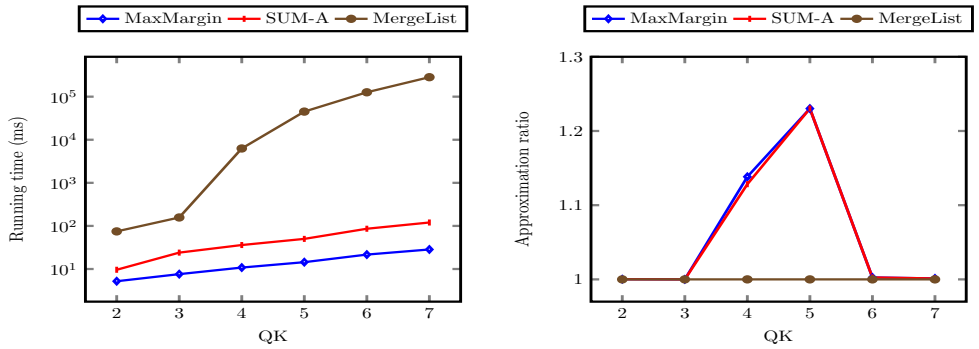
6.3.1. Evaluation on BT

Effect of QK. As shown in Fig. 17(a), MaxMargin and SUM-A outperform MergeList by several orders of magnitude in terms of the response time, especially when *QK* is greater than 3. We notice that, the average number of associated keywords with each object in BT is 3 (see Table 3). When *QK* is greater than 3, MergeList may need to explore much more objects for satisfying the threshold constraint, which



(a) Running time (b) Appro. ratio

Fig. 18: Effect of *TS* on BT



(a) Running time (b) Appro. ratio

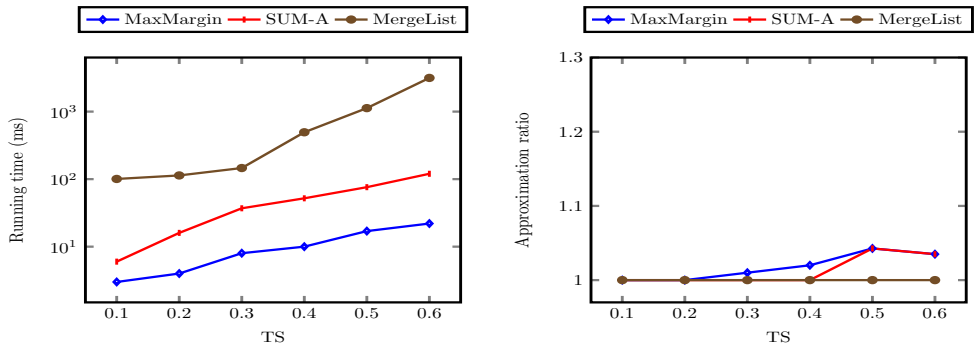
Fig. 19: Effect of *QK* on GN

incurs prohibitive computation cost. In this case, approximate algorithms can prune the search space using LIR-tree. In real datasets, the number of keywords associated with each object is limited, which reduces the number of relevant objects and thus degrades the performance of optimizing strategies used by MaxMargin. As shown in Fig. 17(b), two approximate algorithms perform well in terms of accuracy. Specifically, the approximation ratio of them is close to 1, and they can produce the near-optimal answer in most cases.

Effect of TS. As shown in Fig. 18(a), approximate algorithms are reasonably efficient when *TS* varies from 0.1 to 0.6. However, the response time of MergeList increases dramatically. This is because much more objects in RO_q are required to be explored for finding the answer. With the help of LIR-tree, the response time of approximate algorithms increases almost linearly with *TS*. Fig. 18(b) shows that MaxMargin achieves the same accuracy as SUM-A in most cases, whereas its performance fluctuates. Specifically, when TS=0.2, the approximate algorithms achieve the worst accuracy. As noted earlier, there are five levels, and thus the weight for each level is 0.2 on average. When the threshold is greater than 0.2, more objects are required to satisfy the threshold constraint. This reveals the similar findings as with Fig. 11(b), and thus can be explained with the similar reasons.

6.3.2. Evaluation on GN

Effect of QK. In this experiment, we study the performance of algorithms on GN by varying *QK*. As shown in Fig. 19(a), the response time of approximate algorithms increases almost linearly w.r.t. *QK*, which is similar with the results achieved by BT. This demonstrates the effectiveness of LIR-tree. Also, we observe that all algorithms have much worse response time compared with BT, because the data size of GN is larger



(a) Running time (b) Appro. ratio

Fig. 20: Effect of *TS* on GN

than BT. Fig. 19(b) shows that the accuracy fluctuates when *QK* varies from 3 to 6. This is similar with the observation in Fig. 11(b), and thus can be explained with the same reasons.

Effect of TS. Fig. 20 shows the performance of algorithms as *TS* increases. Specifically, Fig. 20(a) presents similar results with Fig. 18(a). We observe that MergeList has a better response time compared with the results of BT. The reason behind is that, with much more relevant objects in GN, it is more convenient for MergeList to find objects with a greater coverage weight. Fig. 20(b) shows that both two approximate algorithms achieve good accuracy, and return the near-optimal answer in most cases. Besides, the accuracy is more stable compared with that in Fig. 18(b), it is probably because there are much more keywords associated with each object in GN compared with that in BT, and thus can conveniently find the desired object in each of the iterations from the global view.

7. Conclusions and Future Work

In this paper, we study a novel query type called LCSK, which takes into account the keyword level. We prove that the LCSK query is NP-hard by reducing the weighted cover set problem to it. We then propose two algorithms to answering this query exactly and approximately. The exact algorithm, namely MergeList, answers the query by searching the candidate space progressively with two pruning strategies, which is based on the KHT index structure. To be scalable to large datasets, we propose an approximate algorithm called MaxMargin. It finds the answer by traversing down the LIR-tree using the best-first strategy. Moreover, two optimizing strategies, namely the branch and bound strategy and the triggered update strategy are developed to improve the query performance. Extensive experiments on real and synthetic datasets are conducted to evaluate the performance of our proposed algorithms. As verified by the experiments that MaxMargin outperforms the state-of-the-art approximate algorithm, namely SUM-A, by several orders of magnitude with the near-optimal answer.

Future work includes: (1) studying the LCSK query in the dynamic environment with the moving objects; (2) capturing the importance of objects using multiple feature vectors in the LCSK query. Furthermore, we can utilize the spatial keyword search techniques to help the studies in related research fields such as path planning [1, 56], spatial and social information processing and understanding [28, 31, 55], and network information processing [24, 30, 54].

Acknowledgements

We are grateful to anonymous reviewers for their constructive comments on this work. This work was supported by National Program on Key Basic Research Project (973 Program, No.2012CB725305), the NSFC (61428204), the Scientific Innovation Act of STCSM (No.13511504200, 15JC1402400), National Science and

Technology Supporting plan (2015BAH45F01), the public key plan of Zhejiang Province (2014C23005), the cultural relic protection science and technology project of Zhejiang Province.

References

[1] A. Arora and E. M. Scholar. Role of tyrosine kinase inhibitors in cancer therapy. *Journal of Pharmacology and Experimental Therapeutics*, 315(3):971–979, 2005.

[2] C. Bobed and E. Mena. Querygen: Semantic interpretation of keyword queries over heterogeneous information systems. *Information Sciences*, 329:412–433, 2016.

[3] W. Cao, N. Liu, Q. Kong, and H. Feng. Content-based image retrieval using high-dimensional information geometry. *Science China Information Sciences*, 57(7):1–11, 2014.

[4] X. Cao, L. Chen, G. Cong, J. Guan, N.-T. Phan, and X. Xiao. Kors: Keyword-aware optimal route search system. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 1340–1343.

[5] X. Cao, L. Chen, G. Cong, and X. Xiao. Keyword-aware optimal route search. *Proceedings of the VLDB Endowment*, 5(11):1136–1147, 2012.

[6] X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi. Efficient processing of spatial group keyword queries. *ACM Transactions on Database Systems (TODS)*, 40(2):13, 2015.

[7] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 373–384.

[8] X. Cao, G. Cong, C. S. Jensen, and M. L. Yiu. Retrieving regions of interest for user exploration. *Proceedings of the VLDB Endowment*, 7(9):733–744, 2014.

[9] A. Cary, O. Wolfson, and N. Rishe. Efficient and scalable method for processing top-k spatial boolean queries. In *Scientific and Statistical Database Management*, pages 87–95. Springer, 2010.

[10] H. Chen, D. Ni, J. Qin, S. Li, X. Yang, T. Wang, and P. A. Heng. Standard plane localization in fetal ultrasound via domain transferred deep neural networks. *IEEE journal of biomedical and health informatics*, 19(5):1627–1636, 2015.

[11] L. Chen, X. Lin, H. Hu, C. S. Jensen, and J. Xu. Answering why-not questions on spatial keyword top-k queries. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 279–290.

[12] Z. Chen, Y. Liu, R. C.-W. Wong, J. Xiong, G. Mai, and C. Long. Efficient algorithms for optimal location queries in road networks. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 123–134.

[13] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

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

[15] G. Cong, H. Lu, B. C. Ooi, D. Zhang, and M. Zhang. Efficient spatial keyword search in trajectory databases. *arXiv preprint arXiv:1205.2880*, 2012.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[17] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *IEEE 24th International Conference on Data Engineering (ICDE), 2008*, pages 656–665.

[18] K. Deng, X. Li, J. Lu, and X. Zhou. Best keyword cover search. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2015.

[19] Y. Gao, X. Qin, B. Zheng, and G. Chen. Efficient reverse top-k boolean spatial keyword queries on road networks. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2014.

[20] Y. Gao, J. Zhao, B. Zheng, and G. Chen. Efficient collective spatial keyword query processing on road networks. *IEEE Transactions on Intelligent Transportation Systems*, 17(2):1–12, 2015.

[21] T. Guo, X. Cao, and G. Cong. Efficient algorithms for answering the m-closest keywords query. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 405–418.

[22] A. Guttman. R-trees : A dynamic index structure for spatial searching. *SIGMOD*, 14(2):47 – 57, 1984.

[23] J. Hao, H.-F. Leung, and Z. Ming. Multiagent reinforcement social learning toward coordination in cooperative multiagent systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 9(4):20, 2015.

[24] X. Huang, H. Cheng, R.-H. Li, L. Qin, and J. X. Yu. Top-k structural diversity search in large networks. *The VLDB Journal*, 24(3):319–343, 2015.

[25] M. Jiang, A. W.-C. Fu, and R. C.-W. Wong. Exact top-k nearest keyword search in large networks. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 393–404.

[26] Z. Lai, Y. Xu, Q. Chen, J. Yang, and D. Zhang. Multilinear sparse principal component analysis. *IEEE transactions on neural networks and learning systems*, 25(10):1942–1950, 2014.

[27] J. Lee and D.-W. Kim. An effective initialization method for genetic algorithm-based robot path planning using a directed acyclic graph. *Information Sciences*, 332:1–18, 2016.

[28] B. Li, R.-H. Li, I. King, M. R. Lyu, and J. X. Yu. A topic-biased user reputation model in rating systems. *Knowledge and Information Systems*, 44(3):581–607, 2015.

[29] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng. On trip planning queries in spatial databases. In *Advances in Spatial and Temporal Databases*, pages 273–290. Springer, 2005.

[30] R. H. Li and J. X. Yu. Triangle minimization in large networks. *Knowledge and Information Systems*, 45(3):617–643, 2015.

[31] R.-H. Li, J. X. Yu, X. Huang, H. Cheng, and Z. Shang. Measuring the impact of mvc attack in large complex networks. *Information Sciences*, 278:685–702, 2014.

[32] W. Li, J. Guan, and S. Zhou. Efficiently evaluating range-constrained spatial keyword query on road networks. In *Database Systems for Advanced Applications*, pages 283–295. Springer, 2014.

[33] Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang. Ir-tree: An efficient index for geographic document search. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 23(4):585–599, 2011.

[34] X.-H. Lin, Y.-K. Kwok, H. Wang, and N. Xie. A game theoretic approach to balancing energy consumption in heterogeneous wireless sensor networks. *Wireless Communications and Mobile Computing*, 15(1):170–191, 2015.

[35] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu. Collective spatial keyword queries: a distance owner-driven approach. In *Proceedings of the 2013 ACM SIGMOD International conference on Management of data*, pages 689–700.

[36] J. Lu, Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 349–360.

[37] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *Advances in Spatial and Temporal Databases*, pages 205–222. Springer, 2011.

[38] J. B. Rocha-Junior and K. Nørvåg. Top-k spatial keyword queries on road networks. In *Proceedings of the 15th international conference on extending database technology (EDBT)*, pages 168–179. ACM, 2012.

[39] S. Shang, R. Ding, B. Yuan, K. Xie, K. Zheng, and P. Kalnis. User oriented trajectory search for trip recommendation. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT)*, pages 156–167. ACM, 2012.

[40] A. Skovsgaard and C. S. Jensen. Finding top-k relevant groups of spatial web objects. *The VLDB Journal*, 24(4):537–555, 2015.

[41] W.-W. Sun, C.-N. Chen, L. Zhu, Y.-J. Gao, Y.-N. Jing, and Q. Li. On efficient aggregate nearest neighbor query processing in road networks. *Journal of Computer Science and Technology*, 30(4):781–798, 2015.

[42] L. Tan, F. Lin, and H. Wang. Adaptive comprehensive learning bacterial foraging optimization and its application on vehicle routing problem with time windows. *Neurocomputing*, 151:1208–1215, 2015.

[43] Y. Tao and C. Sheng. Fast nearest neighbor search with keywords. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 26(4):878–888, 2014.

[44] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. Ap-tree: Efficiently support continuous spatial-keyword queries over stream. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 1107–1118.

[45] D. Wu, B. Choi, J. Xu, and C. S. Jensen. Authentication of moving top-k spatial keyword queries. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):922–935, 2015.

[46] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen. Joint top-k spatial keyword query processing. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(10):1889–1903, 2012.

[47] L. Xu, Q. Hu, E. Hung, B. Chen, X. Tan, and C. Liao. Large margin clustering on uncertain data by considering probability distribution similarity. *Neurocomputing*, 158:81–89, 2015.

[48] B. Yao, M. Tang, and F. Li. Multi-approximate-keyword routing in gis data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 201–210. ACM, 2011.

[49] Z. Yu, Y. Liu, X. Yu, and K. Q. Pu. Scalable distributed processing of k nearest neighbor queries over moving objects. *IEEE Transactions on Knowledge and Data Engineering*, 27(5):1383–1396, 2015.

[50] C. Zhang, Y. Zhang, W. Zhang, and X. Lin. Inverted linear quadtree: Efficient top k spatial keyword search. *IEEE Transactions on Knowledge and Data Engineering*, 28(7):1706–1721, 2016.

[51] D. Zhang, Y. M. Chee, A. Mondal, A. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *IEEE 25th International Conference on Data Engineering, 2009.*, pages 688–699.

[52] D. Zhang, B. C. Ooi, and A. Tung. Locating mapped resources in web 2.0. In *2010 IEEE 26th International Conference on Data Engineering (ICDE)*, pages 521–532.

[53] D. Zhang, K.-L. Tan, and A. K. Tung. Scalable top-k spatial keyword search. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT)*, pages 359–370. ACM, 2013.

[54] Q. Zhao, S. C. Liew, S. Zhang, and Y. Yu. Distance-based location management utilizing initial position for mobile communication networks. *IEEE Transactions on Mobile Computing*, 15(1):107–120, 2016.

[55] F. Zhou, J. R. Jiao, and B. Lei. A linear threshold-hurdle model for product adoption prediction incorporating social network effects. *Information Sciences*, 307:95–109, 2015.

[56] Z. Zhu, J. Xiao, J.-Q. Li, F. Wang, and Q. Zhang. Global path planning of wheeled robots using multi-objective memetic algorithms. *Integrated Computer-Aided Engineering*, 22(4):387–404, 2015.

[57] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(4):2006, 2006.