Level-aware Collective Spatial Keyword Query

Pengfei Zhang, Huaizhong Lin, Dongming Lu

College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China

Abstract

Spatial keyword queries have received considerable attention recently. Given a location and a set of query keywords, collective spatial keyword query (CoSKQ) retrieves a group of objects, which cover the query keywords collectively and have the minimum cost. However, the increasing importance of *keyword level* (e.g., the level of tourist attraction) for better decision making is ignored by existing works. This observation inspires us to study a generic query paradigm, called *Level-aware Collective Spatial Keyword Query* (LCSK). Specifically, LCSK retrieves a group G of objects to cover query keyword by considering *keyword level*. We prove the NP-hard of this problem by mapping it to the classic Weighted Set Cover (WSC) problem.

To address this problem, we design an exact algorithm with the keyword hash table (KHT) index which organizes the objects in the *flat structure*. Besides, we propose an approximate algorithm with the LIR-tree, which is based on the IR-tree. Based on the LIR-tree, two pruning strategies, namely *branch and bound strategy* and *triggered update strategy* are designed to further improve the response time. Empirical studies over real life and synthetic datasets offer detailed insight into the performance of our proposed algorithms. Specifically, our approximate algorithm runs much faster than the start of the art approximate algorithm with desirable accuracy.

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

1. Introduction

With the development of information technology (e.g., GIS), as well as increasing popularity of services such as Google Earth and Baidu Lvyou, large volumes of geo-textual objects (e.g., facilities, tourist attractions) are available. To handle the query with semantic information, spatial keyword queries [? ? ? ?] are studied extensively recently.

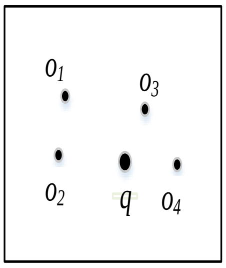
Generally, all existing works on spatial keyword query can be classified into two categories based on the result granularity: (i) some return individual object. Typically, given a location and a set of keywords as arguments, this type of query [? ? ? ?] retrieves objects that cover query keywords individually; (ii) others return a group of objects. In a wide spectrum of applications, however, the user's needs (expressed by keywords) cannot be satisfied by one object individually. To attack this drawback, mCK [? ?], CoSKQ [? ?], BKC [?] and SGK [?] retrieve a group of objects which together cover query keywords. However, few studies take into account the *keyword level* (e.g., the level of tourist attraction, hotel and personal ability). The increasing importance of *keyword level* for better decision making motivates us to investigate novel query.

This paper studies the LCSK query. Specifically, we use the *level vector* to record the levels for corresponding keywords of an object. Besides, we introduce a user-specified *weight vector* to capture the weight user assigned for each level. The LCSK query can be employed in resource scheduling, emergency rescue, etc. Following example illustrates an instance of emergency rescue.

[*]Corresponding author.

[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	10	10
o_2	t_1, t_2, t_4	5,4,1	10	8
o_3	t_1, t_2, t_3, t_4	3,2,5,5	15	6
o_4	t_1, t_2, t_3, t_5	2,3,5,5	15	7

Figure 1: An example of LCSK query

Example 1. Fig. **??** illustrates a simple example of four objects and one query point q. Assume that an earthquake has happened in q. There are four rescue teams $o_1 - o_4$ in the nearby, each of which provides several rescue equipments t_i. The level vector in Fig. **??** records the level of corresponding equipment, which can be used to measure the rescue ability. The *cost* indicates the cost to rescue with this team and *distance* refers to the distance between query point q and o_i.

Several teams are needed to perform the rescue collectively, which is expressed as a query $q = (\ell, \omega, W, \theta)$, where ℓ denotes the location of earthquake and $\omega = (t_1, t_2)$ represents necessary equipments for rescue. The normalized weight vector $W = (0.1, 0.15, 0.2, 0.25, 0.3)$ to measure the rescue ability of corresponding level. For example, the rescue ability of t_1 in o_1 is 0.25 in that the corresponding level is 4. $\theta = 0.5$ denotes the desirable rescue ability, that is to say a group of teams with rescue ability not less than 0.5 are needed. Note that this value can be adjusted flexibly to balance the response time with rescue ability. In this case, we return o_1, o_2 instead of o_3, o_4 which returned by CoSKQ query to perform the rescue.

The reason why o_1, o_2 instead of o_3, o_4 are returned is that: (i) o_3, o_4 do not meet desirable rescue ability when considers the keyword level; 2) we take the cost distance [**?**] as the metric, and o_1, o_2 meet the desirable rescue ability with minimum cost distance.

In general, we enhance existing works from three aspects. First, we take into account keyword level, which is crucial for user to make decision. Second, we introduce weight vector to capture the weight user assigned for levels, which offers greater flexibility for user to make decision. Third, we take the cost distance in [**?**] as our metric. We believe this metric is more realistic by considering both the cost of the object o and distance between o and query q. In view of this, we propose a novel query paradigm LCSK, which paves the way for offering better decision making.

Specifically, given a spatial database O, and a query $q = (\ell, \omega, W, \theta)$, where ℓ represents the location and ω is the query keywords. W is a user-specified weight vector and θ is a threshold. LCSK query retrieves a group G of objects that meet the following two conditions simultaneously:

- For each query keyword t, the sum of coverage weight of G to t is not less than θ;
- The cost distance of G is minimized.

In this work, we map the classic WSC problem to LCSK query, which indicates that LCSK query is NP-hard. To address this problem, we design the exact algorithm MergeList. MergeList handles the query with the flat index structure KHT, which supports the fast access to objects that contain query keywords. MergeList retrieves the optimal result set by constructing the candidate sets with pruning strategies. To further improve the response time, we propose an approximate algorithm MaxMargin with hierarchical index structure LIR-tree, which is based on IR-tree. With LIR-tree, two pruning strategies are devised to enhance the performance. Greedily, MaxMargin picks the current optimal entry in each iteration until the sum of coverage weight not less than $q.\theta$ for all query keywords.

To summarize, we make the following contributions.

- We define a novel query LCSK, which retrieves a group of objects with the keyword level and cost distance function. We prove this problem is NP-hard. To the best of our knowledge, this is the first work to take into account keyword level, which is crucial for better decision making.

- We propose an exact algorithm MergeList with the flat index KHT. Besides, we design an approximate algorithm MaxMargin with the hierarchical index structure LIR-tree, which is based on IR-tree. To further improve response time, *branch and bound strategy* and *triggered update strategy* are utilized by MaxMargin.
- We conduct comprehensive experiments over synthetic and real life datasets to verify the efficiency and effectiveness of the 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 NP-hard complexity of it. We present our exact algorithm MergeList in Section 4 and approximate algorithm MaxMargin in Section 5, respectively. Section 6 gives the empirical study and Section 7 concludes the paper.

2. Related Work

In this section, we mainly overview the existing work related to our LCSK queries, focusing mostly on conventional spatial keyword queries and collective spatial keyword queries.

2.1. Conventional Spatial Keyword Queries

The conventional spatial keyword queries [? ?] take a location and a set of keywords as arguments, and return objects that can satisfy user's needs solely. There are lots of efforts on conventional spatial keyword queries. By combining with existing queries of database community, there are several variants of conventional spatial keyword queries. We will review them as follows.

Combining with top-k queries. By combining with top-k queries, the top-k spatial keyword queries retrieve k objects with the highest ranking scores measured by a ranking function, which takes both location and the relevance of textual descriptions into consideration. To address the top-k spatial keyword queries efficiently, various hybrid indexes have been explored. This branch includes [? ?] (IR-tree), [?] (SKI), [? ?] (S2I). Cong et al. [?] utilizes the hybrid index B^{ck}-tree to facilitate the query processing over trajectory data. Wu et al. [?] handles the joint top-k spatial keyword queries utilizing the W-IR-Tree index. Zhang et al. [?] demonstrates that I^3 index, which adopts the Quadtree structure to hierarchically partition the data space into cells, is superior to IR-tree and S2I. Gao et al. [?] studies the reverse top-k boolean spatial keyword queries on the road network with count tree. Most recently, Chen et al. [?] proposes an index-based method to answer the why not question on spatial top-k keyword query, which offers the flexibility to obtain the result user desired.

Combining with NN queries. The spatial keyword NN queries retrieve object that closes to the query location and contains the query keywords. Several variants have been explored. Tao et al. [?] designs the SI-index to cope with multidimensional data, which eliminates the drawback of IR2-Tree [?] with Z-cuves. Lu et al. [?] studies the RSTkNN query, finding objects that take a specified query object as one of their k most spatial-textual similar objects.

Combining with route queries. The conventional route queries [?] in spatial database search the shortest route that starts at location s, passes through as least one object from each category in C and ends at t. Yao et al. [?] proposes the multi-approximate-keyword routing(MARK) query, which searches for the route with the shortest length such that it covers at least one matching object per keyword with the similarity larger than the corresponding threshold value. The problem of keyword-aware optimal route search (KOR) is studied in [?], to find the route which covers a set of user-specified keywords, a specified budget constraint is satisfied, and an objective score of the route is optimal. Three algorithms are proposed for this problem in [?], and the corresponding system of KOR is provided in their subsequent work [?].

2.2. Collective Keyword Queries

All these works aforementioned return objects that meet user's needs solely. However, in real life applications, it is common to satisfy user's needs collectively by a group of objects. The mCK queries [? ?] return a set of objects to cover the query keywords. However, in the context of mCK, each object associates with only one single keyword and without taking keyword level into consideration. Literature [?] proposes

Notations	Explanations
q	The LCSK query of form:$(\ell, \omega, W, \theta)$
RO_q	The relevant objects to query q of spatial database O
G	The group of objects returned by our algorithms
$\|S\|$	The cardinality of set or vector S
$cost(o)$	The intrinsic value and meanwhile the cost for user to obtain o
$cd(o, q)$	The cost distance of object o to query q
$cw(o, t)$	The coverage weight of t covered by o
$cov(G, q)$	The coverage weight of q covered by group G
KHT	The flat index structure used by exact algorithm
$LIR - tree$	The hierarchical index structure used by approximate algorithm
$cr(e, q)$	The contribution ratio of entry e to q
$dcr_q^r(e)$	The dynamic contribution ratio of e to q when $\|G\| = r$

Table 1: Summary of the notations used.

several efficient approximate algorithms with an exact algorithm to handle the mCK problem. The most similar work to ours is CoSKQ queries [**?** **?** **?**]. With the maximum sum cost function, Cao et al. [**?**] provides approximate algorithms as well as an exact algorithm. To further improve the performance, Long et al. [**?**] proposes a distance owner-driven approach, besides they also propose a new cost measurement called diameter cost and design an exact and an approximate algorithm for it. In [**?**], Cao et al. extends their previous work [**?**] by studying various variants of CoSKQ query with different cost functions.

Although both CoSKQ and our query retrieve a group of objects as result, however, our work differs with CoSKQ significantly in three aspects: (i) we take into account keyword level, which is crucial for user to make decision; (ii) we introduce weight vector to capture the weight user assigned for levels, which offers greater flexibility for user to make decision; (iii) we take the cost distance in [**?**] as our metric. Besides, Deng et al. [**?**] studies the BKC query which considers the keyword rating information and returns best keyword cover, which is different with ours in the query goal. What's more, weight vector is not utilized by BKC.

3. Problem Statement

In this section, we first introduce the fundamental concepts used in this paper. Then we will prove the NP-hard complexity of our problem.

Let O denotes the spatial database. Each object $o \in O$ is associated with a location $o.\ell$, a set of keywords $o.\omega$ to capture the textual description and a $|o.\omega|$ -dimensional level vector $o.\nu$, where $|o.\omega|$ refers to the cardinality of $o.\omega$. The ith element of $o.\nu$ represents the level of corresponding keyword in $o.\omega$. To keep our discussion simple, we consider keyword level as a positive integer only in this paper. And we assume that the level ranges from 1 to 5 in our paper (e.g., there are total 5 levels for tourist attractions). However, the range of level can be extended to any scope straightforward.

Definition 1. (Cost). Given an object $o \in O$, we define the cost of o as the sum of levels of $o.\nu$ which can be denoted as:

$$cost(o) = \sum_{i=1}^{|o.\nu|} o.\nu_i \tag{1}$$

In Equation **??**, $|o.\nu|$ refers to the the cardinality of $o.\nu$. In our work, we utilize the sum of levels as the measurement of the intrinsic value and meanwhile the cost to obtain this object. For instance, the higher the level of the tourist attraction, the higher the cost to visit it in real life.

Definition 2. (Cost Distance). Given a query q and an object $o \in O$, the cost distance of o to q can be denoted as:

$$cd(o, q) = cost(o) \cdot dist(o, q) \tag{2}$$

In Equation **??**, dist(o,q) refers to the Euclidean distance between o and q. Comparing with the cost function used by [? ?], cost distance is more adaptive to real application scenarios in that, it not only takes the distance but also the intrinsic value of o into consideration.

Definition 3. (Object coverage weight). Given a keyword t, a weight vector W and an object $o \in O$. We use the notation $o.\nu^t$ to denote the corresponding keyword level of t in $o.\nu$. Then the weight that t covered by o can be represented as:

$$cw(o, t) = W[o.\nu^t] \tag{3}$$

Differing with CoSKQ in which keyword t either covered by object o or not, in our work we take the coverage weight cw as our measurement. Note that if t is not contained by $o.\omega$, we set $cw(o, t) = 0$. We use $cov(o, q) = \sum_{t \in q.\omega} cw(o, t)$ and $cov(G, t) = \sum_{o \in G} cw(o, t)$ to denote the weight that q is covered by o and the weight that t is covered by a group G of objects, respectively. For example, as depicted in Example 1, we know that $cov(o_1, q) = cw(o_1, t_1) + cw(o_1, t_2) = W[4] + W[5] = 0.25 + 0.3 = 0.55$.

Definition 4. Level-aware Collective Spatial Keyword Query (LCSK Query). The LCSK query $q = (\ell, \omega, W, \theta)$, where ℓ represents the location and ω is the query keywords. W is a user-specified weight vector and θ is a threshold. LCSK query aims at retrieving a group G of objects that collectively satisfy the following two conditions:

- For each keyword $t \in q.\omega$, $cov(G, t) \geq q.\theta$;
- $\underset{G}{\arg\min} \sum_{o \in G} cd(o, q)$.

Given a LCSK query q, we claim an object o is ***relevant*** to q if $o.\omega$ contains at least one keyword $t \in q.\omega$. We use notation RO_q to denote the set of objects that *relevant* to q in O. It is sufficient to take only RO_q instead of O into account for a specific query q. If a group G of objects satisfy the first condition of Definition 4, we say that G is a ***feasible solution*** of query q. Put differently, LCSK query returns the *feasible solution* with minimum cost distance. For ease of reference, Table **??** summarizes the variables used in our paper.

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

PROOF. *To prove the NP-hard complexity of our problem, we reduce the classic Weighted Set Cover (WSC) problem to our LCSK query. Typically, an instance of the WSC problem of the form $< U, S, C >$, where $U = \{1, 2, 3, ..., n\}$ of n elements and a family of sets $S = \{S_1, S_2, S_3, ..., S_m\}$, where $S_i \subseteq U$ and $\cup S_i = U$. For each $S_i \subseteq U$ there is a positive cost $c_i \in C$ to indicate the weight of S_i. The decision problem is to decide if we can find a subset F of S such that $\cup_{S_i \in F} S_i = U$ with the sum of cost of F, e.g., $\sum_{S_i \in F} c_i$ is minimized.*

To reduce the WSC problem to LCSK query q, there are two major steps. First, we construct the query q. We take all the elements of U as the query keywords $q.\omega$. Then, we set the weight vector $q.W = \{1, 0, 0, 0, 0\}$ and the threshold $q.\theta = 1$. Note that, $q.\ell$ can be set arbitrarily without any influence. Second, spatial database O of query q is constructed as follows. We observe that each set S_i corresponds to a spatial object o_i and the associated keywords set $o_i.\omega$ is comprised of elements of S_i. We take the value of c_i as the cost distance $cd(o_i, q)$ and set each level of $o_i.\nu$ to 1. After doing so, it's obvious that there is a solution to weighted set cover problem if and only if there is solution to query q.

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

Table 2: *KHT* entries

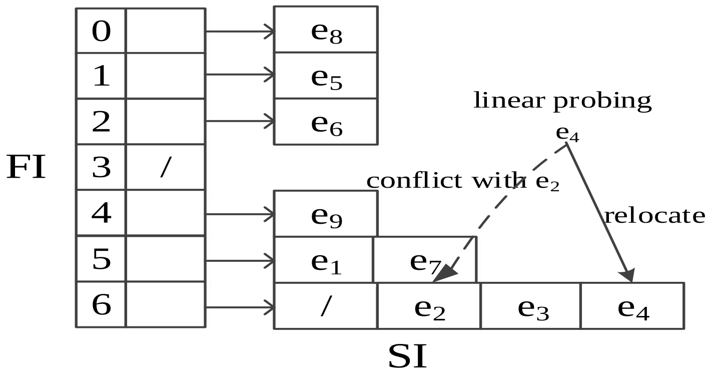


Figure 2: The KHT instance

4. Exact Algorithm

In the context where data size or the number of query keywords is limited, $|RO_q|$ may be small. As mentioned above (Section 3), it's sufficient to answer the query q only with RO_q. Motivated by these observations, we devise an exact algorithm MergeList in this section. Before explaining the algorithm, we first introduce the index used by MergeList.

4.1. KHT Index

In most cases, given a query q, we only need to access a small fraction of objects, that's to say the objects in RO_q. To support fast access to RO_q, we propose the keyword inverted index structure which is organized as a hash table with *perfect hashing technique*. For brevity, we will refer to the index structure as KHT index.

As illustrated in Table **??**, each entry of KHT of the form $(eid, t, olist, fi, si)$, where eid is the identifier of entry and t represents a distinct keyword which followed by a list of objects $olist$ which includes all objects in O that contain t. fi and si correspond to the first index and second index of entry, respectively. Fig. **??** shows an instance of KHT index for entries in Table **??**. Since the limitation of space, we take eid to denote the corresponding entry in Fig. **??**.

To balance the search efficiency with storage space, we combine the two-levels index technique and perfect hashing technique [**?**] in our work. We assign each entry e a two-levels index (fi, si), with which we can retrieve a KHT entry in nearly $\Theta(1)$.

As depicted in Fig. **??**, we determine the index of an entry in two steps. In the first step, we first map the keyword t of entry to the $FTemp$ with the string hash function BKDRHash. Note that, although we take BKDRHash as hash function, other functions can also be applied. Then, we obtain the fi by performing a

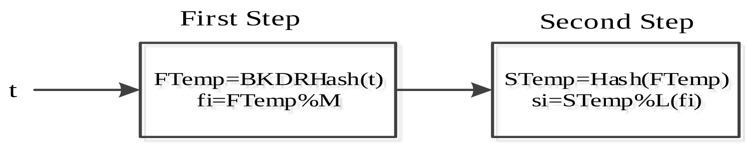


Figure 3: The Hash Function

modulus on $FTemp$ with first-level index length M. In the second step, we hash the $FTemp$ with an integer hash function and then perform a modulus on $STemp$ with second-level index length of fi to get si. After these two steps, each KHT entry has an index (fi, si). Note that, a delicate situation arises when two or more entries share the same index (fi, si) with probability less than 0.5, which is demonstrated in [**?**]. In this case, *"linear probing"* technique is utilized to address this problem. As illustrated in Fig. **??**, there is a conflict when to insert e_4 into KHT, since e_2 has already stayed in location $(6, 1)$. Due to location $(6, 2)$ is occupied as well, with "linear probing", e_4 is inserted into $(6, 3)$ ultimately, just as the solid arrow shown.

From the discussion above, we know that with two-levels index (fi, si), KHT organizes objects in a flat structure which supports the fast access to relevant objects. For a specific query q, we can retrieve the RO_q in nearly $\Theta(|q.\omega|)$ and prune the unnecessary visits significantly.

4.2. The Exact Algorithm MergeList

The straightforward strategy for the exact algorithm is to enumerate all subsets of RO_q and return the one, which covers all query keywords not less than a given threshold and with minimum cost distance, as our optimal solution G (e.g., a group of objects in RO_q). However, this yields an exponential time complexity in terms of the number of objects in RO_q.

To further prune the search space, we delve into several efficient pruning strategies as below.

Firstly, we sort objects of RO_q in ascending order of the cost distance. We record the current optimal solution which is also a feasible solution with COS, and the sum of cost distance of COS, that is to say $\sum_{o \in COS} cd(o, q)$ with $minCost$. Instead of enumerating all subsets of RO_q randomly, we construct the candidate subsets SS by adding object into existing candidate subset ecs progressively. For each $ecs \in SS$, the first condition in Definition 4 has not been satisfied, that is to say ecs is not a feasible solution. In addition, the sum of cost distance of ecs less than $minCost$, e.g., $\sum_{o \in ecs} cd(o, q) < minCost$. By maintaining and updating the SS iteratively, MergeList approaches to the optimal solution progressively.

Lemma 1. *Given a sorted RO_q which in ascending order of the cost distance $cd(o, q)$, where $o \in RO_q$. If for any existing candidate set* ***ecs****, the cost distance sum of ecs and the current visitorial object* ***cvo*** *in RO_q is not less than minCost, e.g., $\sum_{o \in ecs} cd(o, q) + cd(cvo, q) \geq minCost$, then COS is the optimal solution.*

Proof. *Since RO_q is sorted in ascending order of the cost distance $cd(o, q)$, where $o \in RO_q$. We know that each object after cvo in RO_q has a higher cost distance than cvo. If the cost distance sum of any ecs and cvo is not less than minCost, that is to say there is no ecs $\in SS$ can be the optimal solution, so we can terminate our procedure immediately.*

Secondly, there is an ***"Apriori property"*** in the data mining field. ***"All nonempty subsets of a frequent itemset must also be frequent"***. In the sequel, we present a similar pruning strategy.

Lemma 2. *If the cost distance sum of any ecs $\in SS$ and cvo larger than the minCost, we can prune the ecs safely and any superset of it needs not to be computed.*

Proof. *If the cost distance sum of any ecs $\in SS$ and cvo larger than the minCost, we know that ecs cannot be the optimal solution, since there is a current optimal solution COS with minCost. For any superset of ecs, which with higher cost distance neither can be the optimal solution as well, so we can prune them safely.*

Further, only if *ecs* is a feasible solution after adding the current visitorial object *cvo* into it, we can filter out any superset $ecs^{'}$, since *ecs* is superior to $ecs^{'}$ anyway.

In a word, Lemma 1 permits us to terminate the procedure earlier and Lemma 2 provides significant pruning ability.

The MergeList algorithm can be summarized as following three steps.

- Step 1 (Construct RO_q): In this step, we construct the RO_q and sort it in ascending order of the cost distance.
- Step 2 (Add *cvo* into *ecs*): For current visitorial object *cvo* in RO_q, we verify for each *ecs* whether delete it from SS or combine it with *cvo* and put it into SS.
- Step 3 (Iterative step): Repeat Step 2, until the terminal condition in Lemma 1 is met, and return COS as the final solution.

Specifically, we elaborate the MergeList in Algorithm 1.

Algorithm 1: *MergeList*

```
Input  : The $KHT$ and the query $q$.
Output: A group $G$ of objects as resulting solution.
1  $COS \longleftarrow \emptyset$;
2  $SS \longleftarrow \emptyset$;
3  $minCost \longleftarrow$ INFINITE_MAX;
4  $RO_q \longleftarrow$ compute the relevant objects to query $q$ with $KHT$;
5  sort $RO_q$ in ascending order of cost distance;
6  put empty set Ø into SS;
7  for each object $cvo \in RO_q$ do
8      if $cd(cvo, q) \geq minCost$ then
9          break;
10     for each $ecs \in SS$ do
11         if the cost sum of ecs and cvo not less than $minCost$ then
12             delete $ecs$ from $SS$;
13             continue;
14         $tempSet \longleftarrow ecs \cup \{cvo\}$;
15         if $tempSet$ is a feasible solution then
16             $COS \longleftarrow tempSet$;
17             $minCost \longleftarrow$ the cost of $tempSet$;
18             delete $ecs$ from $SS$;
19         else
20             put $tempSet$ into SS;
21     if $SS == \emptyset$ then
22         break;
23 $G \longleftarrow COS$;
24 return $G$ as the final solution;
```

As discussed above, in MergeList algorithm, we construct the existing candidate set *ecs* by adding the *cvo* into it progressively. We use SS to store all the *ecs*, and initiate SS with the empty set (line 6). For each object *cvo* in RO_q, if the cost distance $cd(cvo, q)$ not less than *minCost* then we terminate our procedure (lines 8-9), since COS is already the optimal solution (Lemma 1). Otherwise, for each *ecs* satisfies Lemma 2, we prune it directly (lines 11-13). If *tempSet* is a feasible solution, we use it to update the COS and *minCost* (lines 15-18), otherwise we add it into SS.

OID	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)

SID	Action	SS	minCost	COS
S_0	Initiation	$\emptyset$	$+\infty$	$\emptyset$
S_1	Visit o_1	$\emptyset$,$\{o_1\}$	$+\infty$	$\emptyset$
S_2	Visit o_2	$\emptyset$,$\{o_1\}$,$\{o_2\}$,$\{o_1,o_2\}$	$+\infty$	$\emptyset$
S_3	Visit o_3	$\emptyset$,$\{o_1\}$,$\{o_2\}$,$\{o_1,o_2\}$,$\{o_3\}$,$\{o_1,o_3\}$	6.5	$\{o_2,o_3\}$
S_4	Visit o_4	$\emptyset$,$\{o_4\}$	6.5	$\{o_2,o_3\}$
S_5	Visit o_5	$\emptyset$,$\{o_4\}$	6.5	$\{o_2,o_3\}$

(b)

Figure 4: An example of MergeList

Example 2: Consider a query q with two keywords $q.\omega = \{t_1, t_2\}$. Fig. **??**a illustrates the RO_q, the cost distance to q and the corresponding coverage weight. We illustrate the process of MergeList in Fig. **??**b. There are total six steps to answer this query.

- Step 1 (Initiation): We initiate the SS, $minCost$ and COS in this step.
- Step 2 (Visit o_1): Due to RO_q in Fig. **??**a has been sorted, we visit the object according to the order in Fig. **??**a. We first visit o_1, and merge it with *ecs* in SS.
- Step 3 (Visit o_2): Object o_2 is merged with *ecs* in SS.
- Step 4 (Visit o_3): In this step, we obtain a feasible solution $COS = \{o_2, o_3\}$. Now, we filter *ecs* with Lemma 2.
- Step 5 (Visit o_4): In this step, *ecs* which satisfies Lemma 2 is pruned by COS.
- Step 6 (Visit o_5): Because the cost distance of o_7 is larger than *minCost*, so Lemma 1 is met and COS is the optimal solution.

Theorem 2. *(Correctness of MergeList): The MergeList algorithm always return the correct result set.*

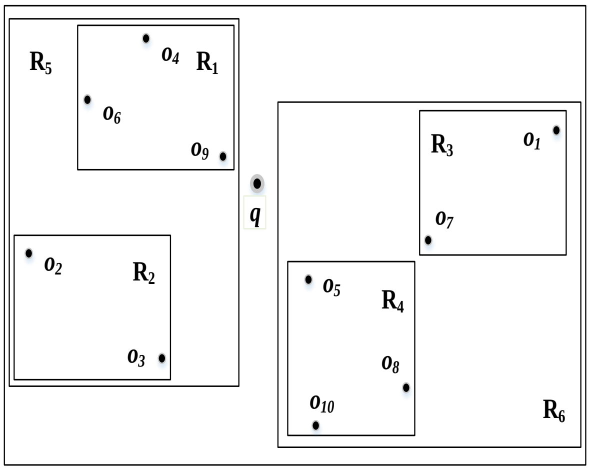
Proof. *Assuming the number of objects in RO_q is n, hence, there are up to $2^n - 1$ non empty existing candidate set ecs inSS. Each ecs either used to update the COS or pruned by the COS. If the cost distance sum of ecs and cvo less than minCost, we take ecs $\cup \{cvo\}$ as our COS. Otherwise, we prune ecs and any superset of it. Hence, it is sufficient to show that MergeList never prune any feasible solution whose cost distance less than minCost (false negatives), and never maintain the ecs whose cost distance larger than minCost (false positives). So the MergeList algorithm always return the correct result.*

5. Approximate Algorithm

MergeList retrieves the RO_q quickly with KHT index. Even though with pruning strategies, however, enumerating all possible subsets is still time consuming especially when the data size is huge. In this Section we propose an approximate algorithm MaxMargin to support query efficiently under various contexts. We show the hierarchical index LIR-tree used by MaxMargin in Section 5.1 and propose the efficient pruning strategies in Section 5.2. Finally, we describe our MaxMargin algorithm in Section 5.3 with theoretical analysis.

5.1. LIR-tree

In the spatial database, the R-tree index [**?**] has been utilized extensively. To support the spatial keyword query efficiently, IR-tree [**?**] index is employed. By combining the R-tree index with inverted file, IR-tree supports the query paradigm which considers the spatial proximity and textual similarity efficiently. Differing with KHT index, which organizes the objects with flat structure, IR-tree utilizes the hierarchical structure to index objects. Contrast to the flat structure, the hierarchical structure is adaptive to big data and can be used to prune effectively. Based on the IR-tree, we propose LIR-tree index structure in our work.



Object descriptions			
object	keywords	level vector	cost
o1	t_1, t_3	3, 1	4
o2	t_1, t_6	1, 1	2
o3	t_2, t_5	5, 2	7
o4	t_3, t_7	2, 3	5
o5	t_4, t_7	2, 4	6
o6	t_3, t_5	5, 3	8
o7	t_2, t_6	1, 2	3
o8	t_5, t_6	1, 5	6
o9	t_1, t_7	3, 2	5
o10	t_6, t_7	4, 3	7

Figure 5: The Spatial Database

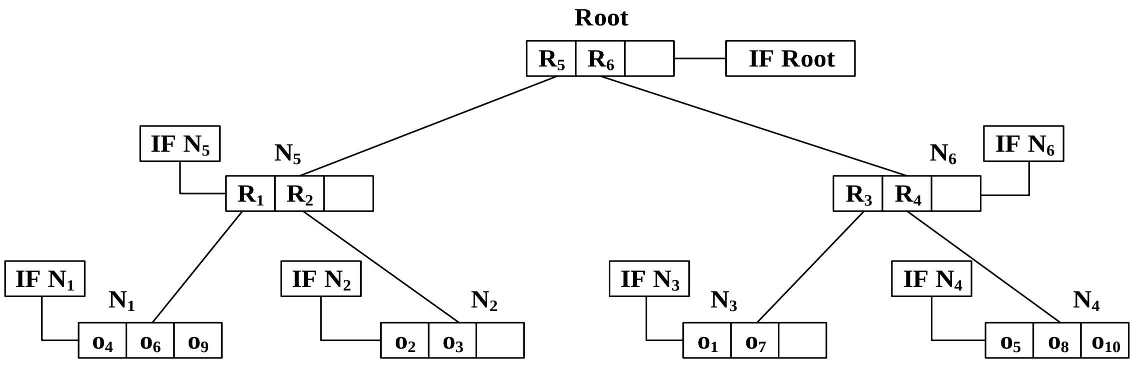


Figure 6: LIR-tree

IF N_1	IF N_2	IF N_3	IF N_4	IF N_5	IF N_6	IF Root
t_1:5;(o_9,3)	t_1:2;(o_2,1)	t_1:4;(o_1,3)	t_4:6;(o_5,2),(o_{10},4)	t_1:2;(N_1,5),(N_2,2)	t_1;4;(N_3,4)	t_1:2;(N_5,2),(N_6,4)
t_3:5;(o_4,2),(o_6,5)	t_2:7;(o_3,5)	t_2:3;(o_7,1)	t_5:6;(o_8,1)	t_2:7;(N_2,7)	t_2:3;(N_3,3)	t_2:3;(N_5,7),(N_6,3)
t_5:8;(o_6,3)	t_5:7;(o_3,2)	t_3:4;(o_1,1)	t_6:6;(o_8,5)	t_3:5;(N_1,5)	t_3:4;(N_3,4)	t_3:4;(N_5,5),(N_6,4)
t_7:5;(o_4,3),(o_9,2)	t_6:2;(o_2,1)	t_6:3;(o_7,2)	t_7:6;(o_5,4),(o_{10},4)	t_5:7;(N_1,8),(N_2,7)	t_4:6;(N_4,6)	t_4:6;(N_6,6)
				t_6:2;(N_2,2)	t_5:6;(N_4,6)	t_5:6;(N_5,7),(N_6,6)
				t_7:5;(N_1,5)	t_6:3;(N_3,3),(N_4,6)	t_6:2;(N_5,2),(N_6,3)
					t_7:6;(N_4,6)	t_7:5;(N_5,5),(N_6,6)

Figure 7: Inverted File

Note that the major difference between LIR-tree and IR-tree results from the inverted file. In the inverted file of IR-tree, there are two components, namely vocabulary table and object list, as follows:

- Vocabulary Table: A vocabulary of all distinct keywords contained by the textual description of an object.
- Object List: For each distinct keyword t, there is a posting list which records all objects whose descriptions contain t.

In LIR-tree index, we modify the object list by adding the level and cost information. As illustrated in Fig. **??**, the object list of the form $< lowerCost, (object, level) >$, where $lowerCost$ represents the lower bound of $cost(o)$ for object o in the object list and $(object, level)$ refers to the object and corresponding keyword level. For instance, in the $IF\ N_4$, for keyword t_4, there are total two objects o_5 and o_{10} in the object list. Due to the cost of o_5 is 6 less than cost 7 of o_{10}, we set the $lowerCost$ as 6. From Fig. **??**, we know that the keyword level of t_4 is 2 for o_5. In the following, we show how to prune with $lowerCost$. Note that, we record the corresponding $lowerCost$ of the child node with $level$ for each non-leaf node, which can be used to enhance the performance furthermore. For instance, in $IF\ N_5$, we record $(N_1, 5)$ in t_1, where 5 denotes the $lowerCost$ of t_1 in N_1.

5.2. Prune Strategy

To address the classic WSC problem, literature [**?**] proposed a greedy strategy, which iteratively selects the current optimal subset and updates subsets have not been accessed yet accordingly. Besides, this strategy is also adopted by the existing works [**? ?**] to answer the spatial keyword query. In literatures [**? ?**], the approximate algorithm performs the query by traversing IR-tree in the top-down manner. Initially, the root node of IR-tree is put into priority queue Q. In each iteration, the top entry e is fetched out. If e is an object then puts e into result set G and then updates remaining entries in queue accordingly. Otherwise, all the child nodes of e are added into queue.

Obviously, two major drawbacks degrade the performance of existing approximate algorithm: (i) lacking of efficient pruning strategies; (ii) updating all remaining entries in each iteration. To further boost the search efficiency, we take the *branch and bound strategy* and *trigged update strategy* to attack the aforementioned drawbacks accordingly. In the sequel, we first take an look at the basic concepts which lay the foundation of these strategies.

Definition 5. (Minimum Cost Distance). Given a query q and a LIR-tree node n, we define the minimum cost distance of n as:

$$mcd_q(n) = minDist(q, n) * lowerCost(q, n) \tag{4}$$

In the above definition, we use $minDist(q, n)$ to represent the possible minimum distance between q and the corresponding MBR of n, and the $lowerCost(q, n)$ represents the minimum cost distance of objects in n that are relevant to q. As illustrated in Fig. **??**, the $lowerCost$ can be obtained from the inverted file directly. For ease of presentation, in the sequel, *we refer to an object o located in the MBR which n represents as o in n.*

Lemma 3. *If there is a feasible solution FS of query q with cost distance FS_{cd}. For any LIR-tree node n, if $mcd_q(n) > FS_{cd}$, then any object o in n can be discarded.*

PROOF. *From the definition of minimum cost distance, we know that for any object o in n, the cost distance $cd(o, q) \geq mcd_q(n)$. Since $mcd_q(n) > FS_{cd}$, then $cd(o, q) > FS_{cd}$ as well. Since FS is already a feasible solution of q, so we can discard all objects in n.*

Branch and Bound Strategy. As Lemma 3 states, we can take the FS_{cd} as the upper bound and prune nodes in the process of traversing LIR-tree. And each time a new object added into the result set G, we refine the value of FS_{cd}, which tightens the upper bound. Obviously, *branch and bound strategy* improves the performance by reducing entries added into priority queue Q.

In the sequel, we introduce the notion and lemma employed by the triggered update strategy.

Definition 6. (Contribution ratio). Given a query q and an entry e, we define the contribution ratio cr of e to q as follows:

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

In the Definition 6, we use $|q.\omega|$ to denote the number of query keywords. By taking both of the coverage weight and the cost distance into account, $cr(e,q)$ can fit in with the real application scenarios better than only use $cov(e,q)$ to evaluate the contribution of e to q. However, as the object added into G, the contribution ratio of entry e decreases accordingly. To capture this changes dynamically, *dynamic contribution ratio* is proposed as below.

Definition 7. (Dynamic contribution ratio). Given a query q and an entry e, we define the dynamic contribution ratio dcr of e when there is r objects in result set G, e.g., $|G| = r$ as follows:

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

In the Definition 7, we use $remain_q^r$ to denote the number of query keywords that the coverage weight has not reach the threshold $q.\theta$. The $cov^r(e,q)$ represents the remaining coverage weight of q covered by object e after r objects are included in G. Note that, both $remain_q^r$ and $cov^r(e,q)$ keep decreasing as object is added into G.

Lemma 4. *Given a query q, for any object o in node n, we know that for any r there is $dcr_q^r(o) \leq dcr_q^r(n)$.*

Proof. *According to the Definition 7, we know that the $cov^r(o,q) \leq remain_q^r * q.\theta$, meanwhile there is $cd(o,q) \geq mcd_q(n)$. Combining these two inequalities we can obtain the inequality $dcr_q^r(o) \leq dcr_q^r(n)$ s-traightforward.*

Lemma 4 provides an upper bound of the dynamic contribution ratio for objects in node n.

Lemma 5. *Given a query q, an entry e, e.g., object or node. Two integers m, n and $m \leq n$, then $dcr_q^n(e) \leq dcr_q^m(e)$.*

Proof. *According to the formula of dynamic contribution ratio, we know that the denominator is fixed all the time. However, the numerator may decrease as object is added into G. That's to say, $cov^n(e,q) \leq cov^m(e,q)$ and $remain_q^n \leq remain_q^m$, which results in the decreasing of dynamic contribution ratio. So, if $m \leq n$, we can safely draw the conclusion that the $dcr_q^n(e)$ is not larger than $dcr_q^m(e)$.*

Triggered Update Strategy. Existing works [**?** **?**] update all remaining entries in queue after adding object into G, which is time consuming and unnecessary. The triggered update strategy not update the entry e until it is popped from priority queue Q. Lemma 5 suggests that if e is still the current optimal entry among the remaining entries in Q, we can add it into G or expend it according to whether e is an object or a node. And all remaining entries in Q need not to be updated, which significantly prunes the updating overhead. As we can see, the *triggered update strategy* improves the performance by reducing the updating overhead.

5.3. MaxMargin Algorithm

With these two pruning strategies, we illustrate our approximate algorithm MaxMargin in the following. Algorithm 2 presents the pseudo-code of MaxMargin.

MaxMargin utilizes the residual vector RV to record the difference between $q.\theta$ and the coverage weight of G for each query keyword dynamically. Initially, each dimension i of RV is set to be $q.\theta$ (lines 3-4). To employ the branch and bound strategy, MaxMargin constructs the feasible solution FS with objects near

Algorithm 2: *MaxMargin*

```
Input  : The query $q$.
Output: A group $G$ of objects as result set.
1  $r \longleftarrow 0$;
2  $\mathsf{G} \longleftarrow \emptyset$;
3  for $i \longleftarrow 0$ to $|q.\omega| - 1$ do
4      $RV[i] \longleftarrow q.\theta$;
5  Construct the feasible solution $FS$ with objects near to $q$;
6  $upBound \longleftarrow$ the cost distance sum of objects in $FS$;
7  $Q.enqueue(LIR.root)$;
8  while $(!Q.empty())$ do
9      $e \longleftarrow Q.dequeue()$;
10     if $e$ is node then
11         recompute the $dcr_q^r(e)$;
12         if $dcr_q^r(e)$ larger than the dcr of top entry in $Q$ then
13             for each child node $cn$ of $e$ do
14                 if $cn$ contains the query keywords and $mdc_q(cn) < upBound$ then
15                     Compute the $dcr_q^r(cn)$;
16                     $Q.enqueue(cn)$;
17         else
18             $Q.enqueue(e)$;
19     else
20         if for each dimension $i$ satisfies $e.cv[i] < RV[i]$ then
21             $\mathsf{G} \longleftarrow G \cup \{e\}$;
22             $refineFS(FS, G)$;
23             $upBound \longleftarrow$ the cost distance sum of objects in $FS$;
24             for $j \longleftarrow 0$ to $|q.\omega| - 1$ do
25                 if $RV[j] \geq e.cv[j]$ then
26                     $RV[j] \longleftarrow RV[j] - e.cv[j]$;
27                 else
28                     $RV[j] \longleftarrow 0$;
29             if for each dimension $j$ satisfies $RV[j] == 0$ then
30                 break;
31             $r{+}{+}$;
32         else
33             for $j \longleftarrow 0$ to $|q.\omega| - 1$ do
34                 if $RV[j] < e.cv[j]$ then
35                     $e.cv[j] \longleftarrow RV[j]$;
36             recompute the $dcr_q^r(e)$;
37             $Q.enqueue(e)$;
38 return $G$ as the result set;
```

to q. Then, we utilize *upBound* to record the upper bound of cost distance of FS. For each entry e of priority queue Q, if e is a node and $dcr_q^r(e)$ larger than the *dynamic contribution ratio* of top entry in Q, that is to say e stills the current optimal entry. Then MaxMargin verifies and enqueues each child node *cn*

Algorithm 3: *refineFS*

```
Input  : The FS and G
1 FS ⟵ $FS \cup G$;
2 sort $FS$ in descending order of cost distance of objects;
3 for each object $o \in FS$ do
4     $Temp \longleftarrow FS - \{o\}$;
5     if $Temp$ is a feasible solution then
6         $FS \longleftarrow Temp$;
```

of it into Q (lines 12-16). For cn with $mdc_q(cn) < upBound$, we discard it directly according to Lemma 3. MaxMargin enqueues e into the queue if it is not the current optimal entry (line 18). If e is an object, there are two different cases. Case 1: If for each dimension i satisfies $e.cv[i] < RV[i]$, it can guarantee that e is still current optimal entry and need not to be updated. MaxMargin adds e into G and then updates RV accordingly (lines 20-31). After adding e into G, we refine the FS and tighten the $upBound$ correspondingly (lines 18-19). We terminate the procedure once the weight coverage constraint for all query keywords is met (lines 29-30). Case 2: Otherwise, we update the RV and then enqueue the updated e into Q.

Algorithm 3 shows the procedure of refining FS. In this procedure, we take a simple strategy. First, we union the existing feasible solution FS with G and sort objects in the descending order of cost distance (lines 1-2). Then, we remove the current object o from FS and verify whether it is still a feasible solution. If it is, then we set it as the new FS (lines 4-6).

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

PROOF. *Inspired by the proof in [?], we provide the approximation ratio proof of MaxMargin here. We use m, n to denote the number of elements in* $q.\omega$ *and* RO_q*, respectively. We define a* $m \times n$ *matrix* $P = (p_{ij})$ *as follows:*

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

In the above formula, we use $q.\omega[i]$ *to denote the ith query keyword and with* $o_j.\omega$ *to represent the keywords associated to* o_j*. According to the definition of P, we know that n columns of P corresponds to n coverage weight vectors. The goal of MaxMargin is to retrieve a group G of objects. And we utilize the incidence vector* $x = (x_j)$ *to denote the cover set which is comprised of a group of objects. Clearly, the incidence vector x of an arbitrary cover satisfies:*

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

From the formula above, we know that the group G of objects which x represented can cover all query keywords with threshold $q.\theta$*. For ease of presentation, in the following, we refer to the cost distance of* o_j *as* c_j*. We use* cov_j^r *to denote the remaining coverage weight of query q covered by object* o_j *before r-th object added into G. And we claim that these inequations imply*

$$\sum_{j=1}^{n} H(\lfloor cov_j^1 + 1 \rfloor) c_j x_j \geq q.\theta \sum_{o_j \in G} c_j \tag{5}$$

for the result set G returned by the greedy heuristic. Once (??) is proved, the theorem will follow by letting x be the incidence vector of an optimal cover.

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

$$\sum_{i=1}^{m} p_{ij} y_i \leq H(\sum_{i=1}^{m} p_{ij}) c_j \quad \textit{for all } j \tag{6}$$

and such that

$$\sum_{i=1}^{m} y_i = \sum_{o_j \in G} c_j \tag{7}$$

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 (??) and (??) have a simple intuitive interpretation: each y_i can be interpreted as the cost distance paid by MaxMargin for covering the keyword $q.\omega[i]$. Without loss of generality, we may assume that G is $\{o_1, o_2, ..., o_r\}$ after r objects added into G, and so

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

for all r and j. If t objects are needed 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 the cov_j^r decreases as the iteration continues (Lemma 5). We assume s is the largest superscript such that $cov_j^s > 0$ then

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

$$
\begin{aligned}
&= c_j \sum_{r=1}^{s} \frac{cov_j^r - cov_j^{r+1}}{cov_j^r} \\
&\leq c_j \sum_{r=1}^{s} \frac{\lfloor cov_j^r + 1 \rfloor - \lfloor cov_j^{r+1} \rfloor}{\lfloor cov_j^r + 1 \rfloor} \\
&= c_j \sum_{r=1}^{s} \sum_{l=\lfloor cov_j^{r+1}+1 \rfloor}^{\lfloor cov_j^r + 1 \rfloor} \frac{1}{cov_j^r} \\
&\leq c_j \sum_{r=1}^{s} \sum_{l=\lfloor cov_j^{r+1}+1 \rfloor}^{\lfloor cov_j^r + 1 \rfloor} \frac{1}{l} \\
&= c_j H(\lfloor cov_j^1 + 1 \rfloor) - c_j H(\lfloor cov_j^s + 1 \rfloor) \\
&\leq c_j H(\lfloor cov_j^1 + 1 \rfloor)
\end{aligned}
$$

Time Complexity. In general, the cardinality of the result set G is at most $s = \frac{|q.\omega| * q.\theta}{min_q(W)}$, where $|q.\omega|$ represents the number of query keywords and $min_q(W)$ denotes the minimum weight in the weight vector of query q. Assume that there is n entries in priority queue Q, in general $n << |RO_q|$ considering the pruning strategies. Given that entry e may be reinserted into queue Q in MaxMargin, in worst case, there is at most $s * n$ times iteration before satisfying the weight coverage constraint for all query keywords. In each iteration, the major overhead results from reinserting e into Q, which is $O(logn)$. As a result, the worst-case time complexity is $O(snlogn)$.

6. Empirical Study

In this section, we experimentally study the performance of our proposed algorithms through comprehensive experiments on both synthetic and real life datasets. We describe the experimental settings in Section 6.1, and report the performance of our algorithms for LCSK query on synthetic and real life datasets in Sections 6.2 and 6.3, respectively.

6.1. Experimental Setup

Algorithms. For the LCSK query, we consider the exact algorithm MergeList in Section 4, which handles the query with flat index KHT. The approximate algorithm proposed in Section 5 with LIR-tree index, which is based on IR-tree. Two efficient pruning strategies are employed to enhance the performance of MaxMargin. To verify the efficiency of MaxMargin, we implement the start of the art approximate algorithm SUM-A which adopted by literatures [**?** **?**] as a comparison. For fair, we modify SUM-A to obey the greedy strategy [**?** **?**] with our proposed LIR-tree index. Besides, we take MergeList as a comparison to evaluate the effectiveness of approximate algorithms.

Data and queries. We conduct experiments with three synthetic datasets and two real life datasets. Experiments are carried out primarily to verify the performance with five major factors, namely, 1) *data size* (DS); 2) *the total number of distinct keywords in the spatial database* (TK); 3) *the upper bound of the number of keywords associated with each object* (KD); 4) *the number of query keywords* (QK); 5) *the weight constraint threshold of query q* (TS). We evaluate their effect on both response time and approximation ratio. Since factors DS, TK and KD are constant for real life datasets, we study DS, TK and KD over synthetic datasets. Note that, for each experiment, e.g., on DS, TK or KD, synthetic data generator generates three types of datasets following the uniform, random and zipf (URZ) distribution respectively as test datasets. We study QK and TS with two real datasets, namely BritishIsles (BT) [**?**] and GN [**?**]. Dataset BT is extracted from OpenStreetMap with a rectangle [(-11.1, 49.6),(2.1,62.5)] to denote the scope. The first coordinate (latitude, longitude) represents the bottom-left corner, and the second coordinate represents the up-right corner. Here, each object is a location with a set of keywords. GN is extracted from the U.S. Board on Geographic Names (geonames. usgs.gov). Each object is associated with a location and a textual description. We randomly generate the level vector for these two real life datasets. Table **??** displays some statistics of the real datasets. And Table **??** illustrates factors studied in our work, and marks their default values in bold. For each experiment, we randomly generate 20 queries with the default values in Table **??**. We report the averaged performance.

All algorithms are 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. The index KHT and LIR-tree are disk-resident and the buffer is set at 4 MB.

6.2. Results on Synthetic Dataset

To measure the comprehensive performance of our algorithms on different types of datasets, in this section, we employ the averaged response time ($URZA\_T$) and approximation ratio ($URZA\_R$) as the measurements, where $URZA\_T = \frac{T_u+T_r+T_z}{3}, URZA\_R = \frac{R_u+R_r+R_z}{3}$. Tu, Tr, Tz and Ru, Rr, Rz represent the response time and approximation ratio of uniform, random and zipf datasets, respectively.

Property	BritishIsles	GN
#objects	298346	977302
#keywords	748162	5286498
#distinct keywords	58367	116466

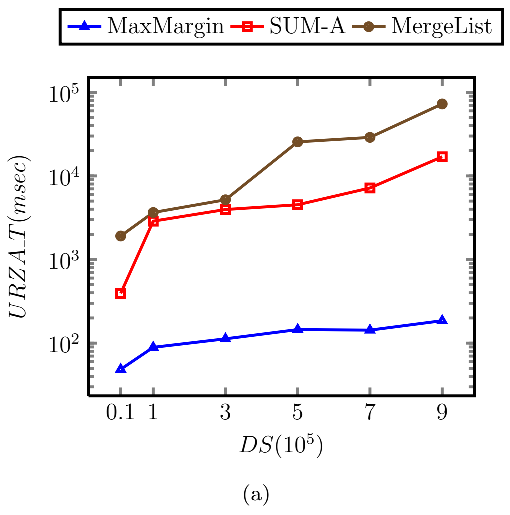
Table 3: The property of real life datasets

Factors	Instance Value
DS(10^5)	0.1,**1**,3,5,7,9
TK	50,100,150,200,250,**300**
KD	3,5,**7**,9,11,13
QK	1,2,**3**,4,5
TS	0.1,0.2,**0.3**,0.4,0.5

Table 4: The query factors

Varying the factor DS. Figs. **??**a and **??**b show the response time and approximation ratio of three algorithms respectively, when we vary the value of DS. In this experiment, we vary the value of DS from 10,000 to 900,000. Note that the y-axis is in logarithmic scale. As expected, the response time of all algorithms increase with the DS. This is because the increasing number of relevant objects $|RO_q|$. However, as shown in the Fig. **??**a, MaxMargin scales much better against the DS than other two algorithms. Even thought SUM-A utilizes LIR-tree index as MaxMargin, however, MaxMargin runs orders of magnitudes faster than SUM-A. The reason for the difference is that MaxMargin applies two pruning strategies, namely *branch and bound strategy* and *triggered update strategy*. The result denotes the efficiency of these two strategies. In Fig. **??**b, we set the approximation ratio of the exact algorithm MergeList equals to 1, which is also adopted by the subsequent experiments. Although both two approximate algorithms SUM-A and MaxMargin are based on the greedy strategy in [**?**], Fig. **??**b shows that SUM-A achieves a little better performance than MaxMargin in terms of accuracy. The difference might result from the different strategy utilized to select optimal entry.

Varying the factor TK. Figs. **??**a and **??**b show the response time and approximation ratio of three algorithms respectively, when we vary the value of TK. As can be seen from Fig. **??**a, the response time of all algorithms decreases as TK increases. The reason behind is that, when other factors are fixed, the larger the TK, the smaller number of relevant objects $|RO_q|$ and thus smaller objects need to be handled during algorithms execution. Specifically, MergeList drops quickly before TK takes the value of 150, and then decreases slightly. This might be because TK has little effect on the number of relevant objects $|RO_q|$



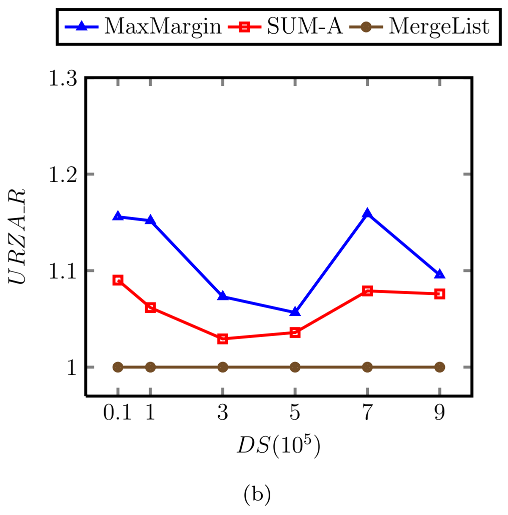
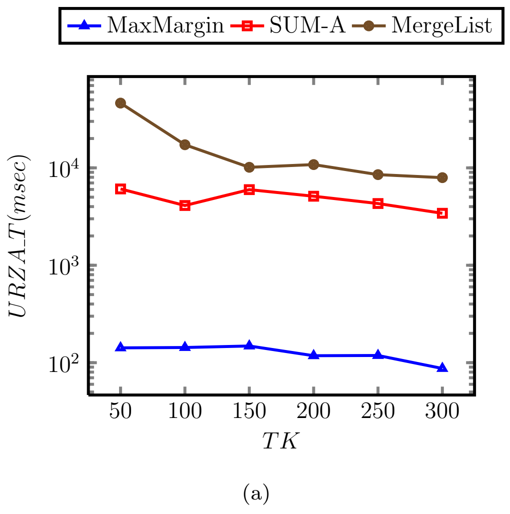


Figure 8: Varying the factor DS



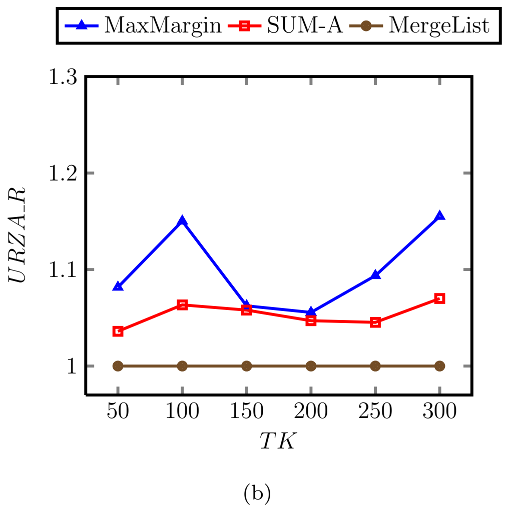


Figure 9: Varying the factor TK

when TK larger than 150. It can be observed from Fig. **??**b that, both two approximate algorithms achieve a good accuracy with an upper bound 1.2, and they can obtain nearly optimal result set.

Varying the factor KD. Figs. **??**a and **??**b show the response time and approximation ratio of three algorithms respectively, when we vary the value of KD. Differing with the method in [**?**], which fixes the number of keywords associated with objects, in this experiment, we set the upper bound for the number of keywords associated with objects to KD. For each object, k keywords are generated, where k is a positive integer less than the upper bound KD. We vary KD from 3 to 13. Fig. **??**a shows that the response time of all algorithms increases almost linearly with KD. It is understandable that the response time is proportional to the number of relevant objects RO_q, which is also generally proportional to the KD. From Fig. **??**b we know that, although MaxMargin changes slightly in terms of accuracy, however, both two approximate algorithms perform well in terms of accuracy.

The performance on Different Datasets. To further depict the performance of our algorithms

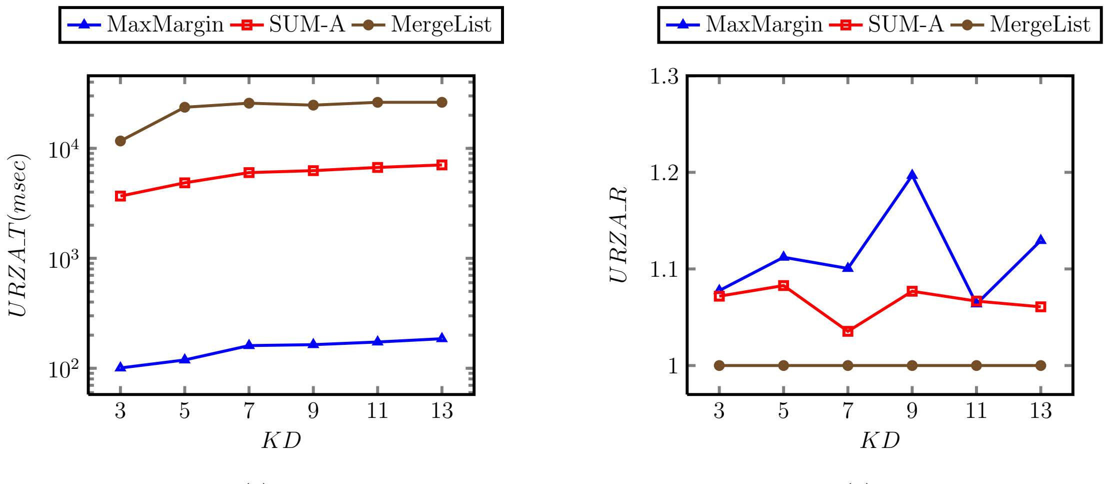


Figure 10: Varying the factor KD

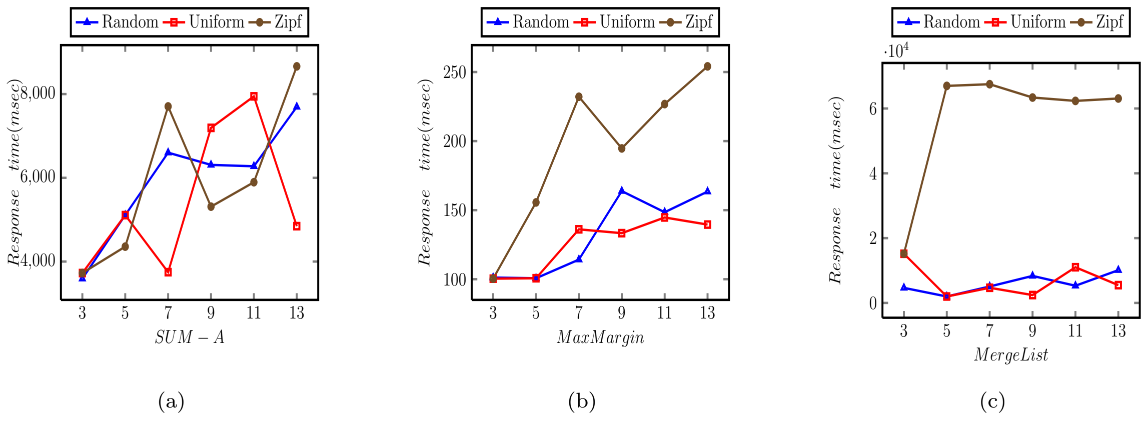


Figure 11: Response Time on Different Datasets

on different types of datasets. In this experiment, we study the performance of algorithms on different datasets (e.g., random, uniform and zipf) from the aspect of KD. Figs. **??** and **??** show the response time and approximation ratio of our algorithms on different datasets, respectively. Specifically, Figs. **??**a, **??**b and **??**c illustrate the response time of SUM-A, MaxMargin and MergeList respectively. Curves in figures correspond to three datasets, namely, uniform, random and zipf. As can be observed from Fig. **??**, similar with the result on KD, the response time of all algorithms increases as KD increases. All algorithms are sensitive to zipf dataset. This is because the distribution of zipf dataset has much effect on the pruning ability. SUM-A performs worse than other two algorithms in terms of stability. This is because the pruning ability provided only with LIR-index, which relies much on the data distribution. Figs. **??**a, **??**b and **??**c illustrate the approximation ratio of SUM-A, MaxMargin and MergeList respectively. Curves in figures correspond to three datasets, namely, uniform, random and zipf. We notice that all algorithms achieve better accuracy on zipf dataset than other two datasets. This might be because that more time (see Fig. **??**) is spent on zipf dataset to find the optimal solution. Uniform dataset performs better than random dataset in terms of accuracy and stability. The instability for random dataset in Fig. **??**b may due to the feature of random data.

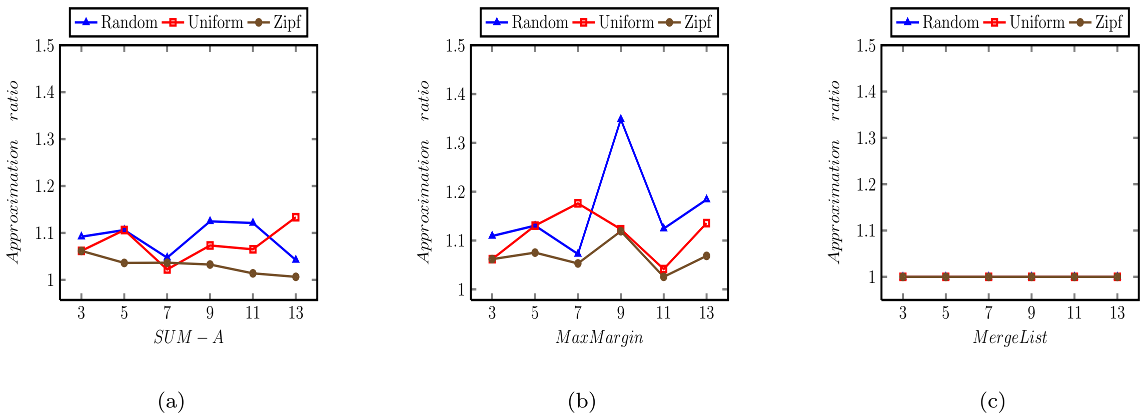


Figure 12: Approximation Ratio on Different Datasets

6.3. *Results on Real Life Dataset*

In this section, we mainly study the performance of our proposed algorithms on real life dataset BT and GN with two factors QK and TS.

6.3.1. *Results on BT*

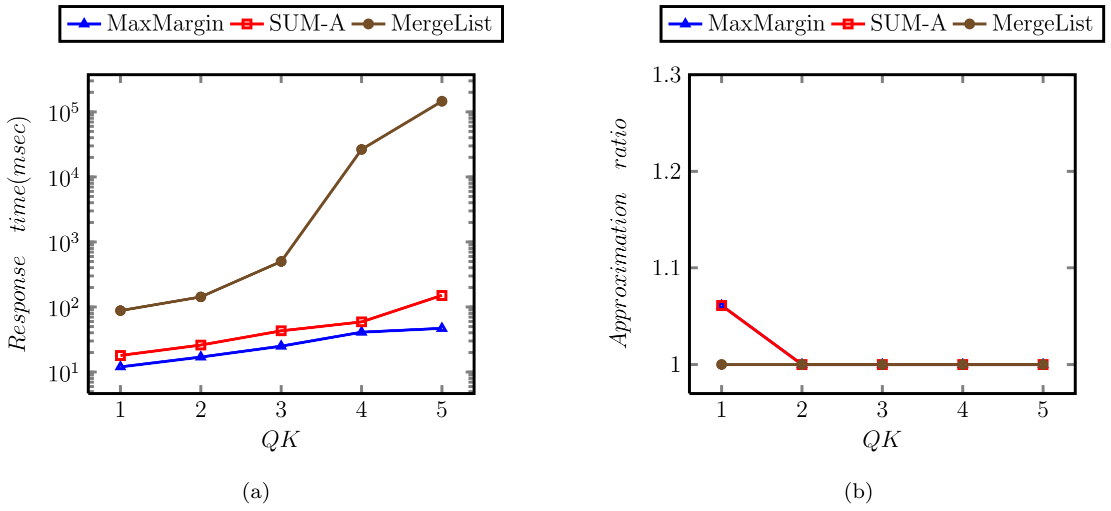


Figure 13: Varying the factor QK

Varying the factor QK. Figs. **??**a and **??**b show the response time and approximation ratio of three algorithms respectively, when we vary the value of QK. In this experiment, we vary the value of QK from 1 to 5. As shown in Fig. **??**a, MaxMargin and SUM-A run much faster than MergeList, especially when the number of QK larger than 3. As can be seen from Table **??**, the averaged number of associated keywords for each object of BT is 3. For MergeList it may need to search much more objects to meet the threshold constraint when QK larger than 3, which is time consuming. In contrast, approximate algorithms can prune unnecessary visits significantly. Note that, in real life dataset, the number of keywords associated to each object is limited, which results in the reducing of relevant objects and then degrades the performance of our pruning strategies utilized by MaxMargin. In this situation, MaxMargin generally runs 2-4 times faster than SUM-A. As can be observed from Fig. **??**b, both two approximate algorithms perform well in terms of accuracy. The approximation ratio of them is very close to 1, and it can always obtain optimal result set.

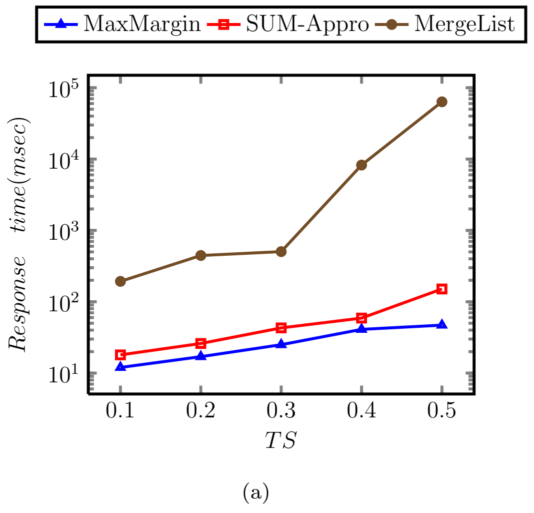


Figure 14: Varying the factor TS

Varying the factor TS. Figs. **??**a and **??**b show the response time and approximation ratio of three algorithms respectively, when we vary the value of TS. In this experiment, we vary TS from 0.1 to 0.5. Generally, the value of TS has much effect on the number of objects accessed. As can be seen from Fig. **??**a that the response time of MergeList increases dramatically. This is because more objects in RO_q are needed to appended into the candidate sets SS, which incurs enormous overhead. In contrast, MaxMargin and SUM-A are more adaptive to TS with LIR-tree index. Fig. **??**b shows that, MaxMargin achieves the same approximation ratio as SUM-A, which is close to the exact solution.

6.3.2. Results on GN

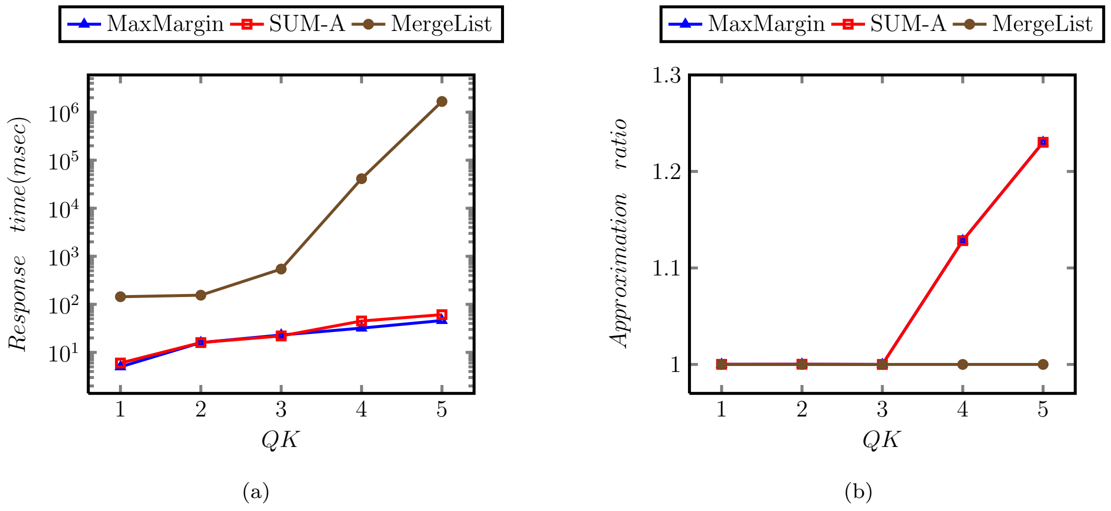


Figure 15: Varying the factor QK

Varying the factor QK. Figs. **??**a and **??**b show the response time and approximation ratio of three algorithms respectively, when we vary the value of QK. As can be seen from Fig. **??**a, the response time of the approximate algorithms SUM-A and MaxMargin increase almost linearly with the number of query keywords, which is similar with the result on BT. This demonstrates the usefulness of LIR-tree based pruning strategies. MergeList increases much faster than the result on BT, since the data size of GN larger than BT. As can be observed from Fig. **??**b, the approximation ratio of two algorithms increase when QK larger than 3, this might be because the increasing of error for approximate to select the current optimal entry.

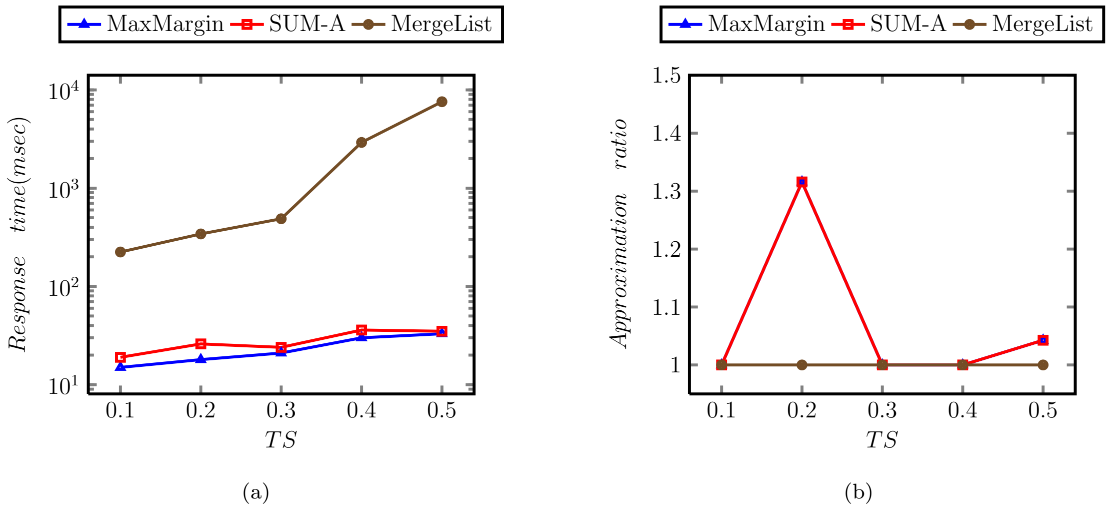


Figure 16: Varying the factor TS

Varying the factor TS. Figs. **??**a and **??**b show the response time and approximation ratio of three algorithms respectively, when we vary the value of TS. Compared with result of Fig. **??**a, Fig. **??**a shows that all algorithms run faster. The reason behind is that, with much more relevant objects in GN, it's more convenient for algorithms to find objects with higher coverage weight. Thus, the response time is reduced. Fig. **??**b presents similar result with Fig. **??**b and can be explained with the same reason.

7. Conclusions and Future Work

In this paper, we introduce the novel query type, namely, LCSK query. By taking into account keyword level, LCSK can support better decision making. We map LCSK to classic Weighted Set Cover (WSC) problem, which means that LCSK is NP-hard. To address this novel problem, we propose an exact algorithm MergeList with the flat index structure KHT, which supports fast access to relevant objects. To further improve query efficiency, we design an approximate algorithm MaxMargin with LIR-tree index structure, which is based on the IR-tree. With two pruning strategies, MaxMargin runs much faster than the start of the art approximate algorithm SUM-A. Extensive experiments on both real life and synthetic datasets are conducted to verify the performance of our proposed algorithms.

In the future work, there are several interesting research directions. One is to research the LCSK problem in the road network scenario. It is also interesting to study other forms of cost function for this problem.

Acknowledgements: We are grateful to anonymous reviewers for their constructive comments on this work. This work was supported in part by the National Science and Technology Supporting Plan-Project no. 2012BAH70F02 and 2013BAH62F01, and the cultural relic protection science and technology project of Zhejiang Province.