Computer Science and Information Systems 18(3):1077–1100 https://doi.org/10.2298/CSIS200131034C

Time-aware Collective Spatial Keyword Query

Zijun Chen[1,2⋆] Tingting Zhao[1] and Wenyuan Liu[1,2]

[1] School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China
[2] The Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Qinhuangdao 066004, China
zjchen@ysu.edu.cn, tingtingzhao@stumail.ysu.edu.cn, wyliu@ysu.edu.cn

Abstract. The collective spatial keyword query is a hot research topic in the database community in recent years, which considers both the positional relevance to the query location and textual relevance to the query keywords. However, in real life, the temporal information of object is not always valid. Based on this, we define a new query, namely time-aware collective spatial keyword query (TCoSKQ), which considers the positional relevance, textual relevance, and temporal relevance between objects and query at the same time. Two evaluation functions are defined to meet different needs of users, for each of which we propose an algorithm. Effective pruning strategies are proposed to improve query efficiency based on the two algorithms. Finally, the experimental results show that the proposed algorithms are efficient and scalable.

Keywords: Collection of objects, TR-tree, Valid time of the objects, Keyword query.

1. Introduction

As textual information on location-based geographic information has been paid more and more attention, spatial keyword query technology is proposed [8]. With the continuous development of spatial keyword query, in some practical applications, people begin to pay attention to the valid time of geo-spatial objects. For example, visitors want to plan a trip based on the opening time of geo-spatial objects, which is the time-aware spatial keyword query [5]. However, [5] only considers the case that a single object contains all the query keywords. As far as we know, there is no work to consider the valid time of objects based on the collective spatial keyword query. The collective spatial keyword query refers to finding a group of objects that together contain all of the query keywords and are near to the query location. Therefore, this paper proposes a new query, i.e., time-aware collective spatial keyword query (TCoSKQ). An example is illustrated in Fig. 1, where a user (at the location of q) plans to visit a "gym" from 7:00 to 8:30, and a "restaurant" from 9:00 to 10:00. Collective spatial keyword query may return $\{o_1, o_2\}$, while TCoSKQ may return $\{o_3, o_2\}$. The reason of the difference is that TCoSKQ considers the valid time of the objects. Although [6] is closely related to ours, they do not consider the case that there exists a query point. [21] considers the valid time of the objects too, but it aims to find a set of k objects ranked the highest according to a ranking function.

⋆ Corresponding author

TCoSKQ considers the positional relevance, textual relevance and temporal relevance between the objects and the query at the same time. In order to solve the TCoSKQ problem, we define two evaluation functions to meet different needs of users, i.e., $score_1$ and $score_2$, which are suitable for the cases that the object containing the query keyword is close to and far from the query position, respectively. Then, TCoA1 algorithm and TCoA2 algorithm are designed for the two evaluation functions, respectively. Both algorithms use multiple Time-aware R-tree (TR-tree) [6] to index the valid time information and the spatial information of objects, that is, objects containing the same keyword are on the same TR-tree. TCoA1 algorithm finds feasible solutions by the distance dominators, which are searched from near to far, starting from the query location. Finally, the feasible solution with the largest $score_1$ is the final result. In order to improve the efficiency of the query, we propose effective pruning strategies for pruning the distance dominators that are unlikely to appear in the final result. TCoA2 algorithm searches for the center object from near to far from the query location, in which TCoA1* algorithm is called to find the feasible solution. TCoA1* algorithm is got by modifying TCoA1 algorithm. The feasible solution with the largest $score_2$ is the optimal solution, and effective pruning strategies are proposed to improve the query efficiency.

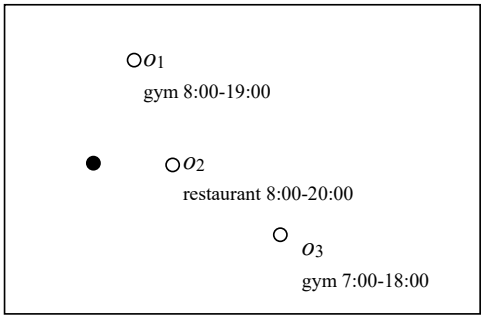


Fig. 1. Example of a TCoSKQ

To summarize, the main contributions of this paper are:

(1) We propose a new query, i.e., time-aware collective spatial keyword query (TCoSKQ).

(2) We propose two evaluation functions (i.e., $score_1$ and $score_2$), and propose effective pruning strategies and algorithms (i.e., TCoA1 and TCoA2) for these two functions to solve TCoSKQ.

(3) We conduct extensive experiments using the data sets to demonstrate the efficiency and scalability of our algorithms.

This paper introduces related work in Section 2. Section 3 gives problem definition. Section 4 gives the TR-tree indexing structure. Section 5 and 6 elaborates TCoA1 and TCoA2 algorithm, respectively. Section 7 gives the experimental results and analysis. Section 8 concludes the paper.

2. Related Work

In recent years, spatial keyword query has attracted much attention from spatial database community. Scholars have proposed effective techniques to deal with spatial keyword query. The early spatial keyword query is mainly for the case of retrieving a single object containing all query keywords and close to the query position. It is roughly divided into three types: Boolean kNN query [2], [18], ranked kNN query [14] and Boolean range query [19].

As people's needs increase, [1] observed that there may be situations where a single object cannot meet the needs of users, therefore, they first proposed collective spatial keyword query (CoSKQ). CoSKQ refers to retrieving a group of spatial web objects such that the group's keywords cover the query's keywords and such that objects are nearest to the query location and have the lowest inter-object distances. Based on two types of cost function, they study two instances of this problem, both of which are NP-complete. So they proposed effective approximate algorithms and exact algorithms to solve the two instances. Based on the shortcomings of the query in [1], [15] defined a new cost function describing the quality of set, proposed a new collective query processing method based on spatial keyword, and devised the corresponding approximate algorithm and exact algorithm.

With the intensive study of the collective spatial keyword query, some variants of the collective spatial keyword query have been proposed. [7] focused on specific spatial keyword set search in specific direction, and proposed a query algorithm based on grid index. [3] proposed an inherent-cost aware collective spatial keyword query, which takes into account the inherent cost of each object, and gave an exact algorithm and approximate algorithm that can solve the problem. Considering the importance of keyword level, [20] proposed a level-aware collective spatial keyword query, the corresponding cost function, the exact algorithm, and the approximate algorithm. Group-based collective keyword querying was proposed in [17], which considers the case of group of users. The query aims to find a region containing a set of objects that covers all the query keywords and these objects are close to the group of users and are close to each other. [9] and [22] have successively proposed methods for solving collective spatial keyword query on the road network. [11] considered the problem of scalable collective spatial keyword queries and proposed a distributed method to solve this problem effectively. In [4], a unified cost function and a unified method are proposed for the collective spatial keyword query problem to systematically solve the query problem.

With the in-depth study of spatial keyword queries, the valid time information of objects has attracted people's attention. [5] proposed a time-aware Boolean spatial keyword query (TABSKQ), which returns the k objects that satisfy users' spatio-temporal description and textual constraint, designed TA-tree index structure, and proposed algorithms to process TABSKQ efficiently. [21] proposed time-aware spatial keyword queries on road networks, which finds the k objects satisfying users' spatio-temporal description and textual constraint, and devised several effective algorithms using the TG index. [6] proposed a time-aware spatial keyword cover query (TSKCQ), devised a TR-tree for indexing the temporal information and the spatial information of objects, and proposed an exact algorithm to tackle TSKCQ.

As far as we know, the previous work on collective spatial keyword query does not consider the importance of the object's valid time information. Therefore, we propose

a new query, that is, time-aware collective spatial keyword query, which considers the object's valid time based on the collective spatial keyword query.

3. Problem Definition

In spatial dataset, each object may be associated with one or multiple keywords. We convert the object with multiple keywords into multiple objects in the same location. For any object o with m ($m > 1$) keywords, we will create $m - 1$ other objects with the same location of o so that each of the m objects has only one different keyword. Let D be a set of objects. Each $o \in D$ is associated with a location denoted by $o.\lambda$, a keyword denoted by $o.k$ and a valid time denoted by $o.t$, where $o.t$ is in the form of (st, et) with st and et being the starting time stamp and the ending time stamp of o, respectively. Similar to [5] and [6], we integerize st and et of o, and take one hour as a time unit. As an example, any time stamp among [13:00, 14:00) can be converted to a time unit 13. (13:10, 18:00) can be converted to (13, 18). For simplicity, let et, $st \in [0, 24]$ and $st \leq et$. The input of a time-aware collective spatial keyword query (TCoSKQ) is $q = (\lambda, K, T)$, where $q.\lambda$ is the query location, $K = \{k_1, ..., k_m\}$, $T = \{t_1, ..., t_m\}$, $k_i (1 \leq i \leq m)$ is the ith query keyword, and t_i is a time interval specified in the query for k_i. For the query q, the object set $S = \{o_1, ..., o_m\} (S \subseteq D)$ containing all the keywords in K is called the feasible solution.

Definition 1. *(Time-aware collective spatial keyword query (TCoSKQ)) Given a spatial database D and a query $q = (\lambda, K, T)$, TCoSKQ returns an optimal solution S (which is also a feasible solution), such that $score(q, S) \geq score(q, S')$, where S' is any feasible solution.*

According to the two different needs of users, we give the definition of $score(q, S)$:

$$score(q, S) = \begin{cases} score_1(q, S), & \text{the first evaluation function} \\ score_2(q, S), & \text{the second evaluation function} \end{cases} \tag{1}$$

where $score_1$ and $score_2$ are evaluation functions that satisfy the user's first and second requirement, respectively.

(a) The $score_1$ is defined as:

$$score_1(q, S) = \alpha(1 - \max_{o \in S} \frac{dist(q, o)}{max\_dist}) + (1 - \alpha) \min_{o \in S, k_i \in K, o.k = k_i} \frac{|t_i \cap o.t|}{|t_i|} \tag{2}$$

where $dist(q, o)$ is the Euclidean distance between q and o, max_dist is the maximum distance between any two objects in the spatial database D, and $\alpha (0 < \alpha < 1)$ is a specified parameter. Let $\Gamma_o = \frac{|t_i \cap o.t|}{|t_i|}$, where the object o contains the keyword k_i in K.

(b) The $score_2$ is defined as:

$$score_2(q, S) = \max_{o \in S} score(q, S, o) \tag{3}$$

$$score(q, S, o) = \beta(1 - \frac{dist(q, o)}{max\_dist}) + (1 - \beta) score_1(q', S), o \in S \tag{4}$$

where $q' = (o.\lambda, q.K, q.T)$. According to (3), a function value is obtained by centering on each object in S, so that such an object is called the center object of S. We call o' the best center object if $o' = \arg\max_{o \in S} score(q, S, o)$.

For $score_1$ and $score_2$, the user could choose the function according to the query location. If the user is in the downtown area at present, $score_1$ may be chosen to find the result. But if the user is far away from downtown now, the user may have to choose $score_2$ to find the result.

The notations used in this work are summarized in Table 1.

Table 1. Symbols and Description

Notation	Description
D	a set of objects
Γ_o	the temporal overlap ratio of the object o
$score_1$	the evaluation function that satisfies the user's first requirement
$score_2$	the evaluation function that satisfies the user's second requirement
TR_{k_i}-tree	the time-aware R-tree for keyword k_i
$d(q, elem)$	the distance (or minimum distance) from query q to object (or node) $elem$
$C(q, r)$	a circle with q as the center and r as the radius
$NN(q, k)$	k-keyword nearest neighbor of q

4. TR-tree Index Structure

To process TCoSKQ, we use TR-tree [6] as the index structure of the algorithms, which is an extension of R-tree [10]. On the basis of R-tree, a new dimension is added for indexing valid time of objects.

As far as we know, the current collective spatial keyword queries use the index structure of a single tree, that is, a tree indexes objects in all geographic locations. A single tree structure suits the situation that most keywords are query keywords. However, in practice, users will only use a small fraction of keywords as query keywords. So, multiple trees are also used in this work, one for each keyword. The TR-tree for keyword k_i is denoted as TR_{k_i}-tree.

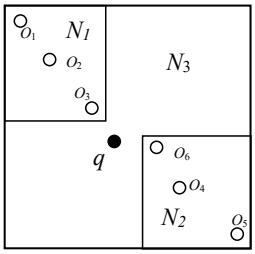
In the case of knowing the query keywords, we only need to find the TR-tree corresponding to the query keyword, which greatly reduces the number of objects to be considered in the query.

Next, we introduce the non-leaf nodes and leaf nodes of TR-tree in detail:

Non-leaf nodes of TR-tree contain entries of the form $N(ptrs, mbr, UT)$, where $ptrs$ is the address of a child node of N, mbr is the minimum bounding rectangle (MBR) of all rectangles in entries of the child node, and UT is a set of the time intervals that are the union of the valid times of the objects in N.

Leaf nodes of TR-tree contain entries of the form $o(id, l, t)$, where id refers to the object o, l represents the coordinates of o, and t is the valid time of o.

Fig. 2a gives the placement of objects containing the keyword "restaurant". Objects and valid time of objects are shown in Fig. 2b. The TR_{k_1}-tree for keyword "restaurant" is created, and the nodes and their corresponding valid time of the tree are shown in Fig. 2c.



(a) Object placement

Object	Valid time
o_1	(6:00, 10:00)
o_2	(7:00, 9:00)
o_3	(7:00, 11:00)
o_4	(13:00, 17:00)
o_5	(21:00, 24:00)
o_6	(9:00, 14:00)

(b) Objects and valid time of objects

Node	Valid time
N_1	(6:00, 11:00)
N_2	(9:00, 17:00), (21:00, 24:00)
N_3	(6:00, 17:00), (21:00, 24:00)

(c) Nodes and time intervals of nodes

Fig. 2. Example of a TR-tree

TR-tree inherits the important features of R-tree: the minimum distance from the query to the parent node is less than or equal to the minimum distance to the child node.

Example 1. In Fig. 2, we assume that the time interval specified in the query for the query keyword "restaurant" is (8:00, 19:00). Thus, we have $\Gamma_{o_1} = 2/11$, $\Gamma_{o_2} = 1/11$, $\Gamma_{o_3} = 3/11$, $\Gamma_{o_4} = 4/11$, $\Gamma_{o_5} = 0$, $\Gamma_{o_6} = 5/11$.

Definition 2. *(MinDist Distance [16]) In Euclidean space of dimension n, the minimum distance between a point q and MBR $N(s,u)$ is denoted by $MinDist(q, N(s,u))$, which is defined as follows:*

$$MinDist(q,N) = \sum_{i=1}^{n} |q_i - r_i|^2, r_i = \begin{cases} s_i, & q_i < s_i \\ u_i, & q_i > u_i \\ q_i, & \text{otherwise} \end{cases} \tag{5}$$

Given a query q,

$$Dist(q,elem) = \begin{cases} dist(q,elem), & elem \text{ is an object} \\ MinDist(q,elem), & elem \text{ is a node} \end{cases} \tag{6}$$

$$d(q,elem) = \frac{Dist(q,elem)}{max\_dist} \tag{7}$$

5. TCoA1 algorithm

We propose TCoA1 algorithm for the evaluation function $score_1(q, S)$ to solve the TCoSKQ problem. Before we introduce TCoA1 algorithm, we first introduce some notations.

Given a query $q = (\lambda, K, T)$, let S be a solution that satisfies the query, then

(1) The distance dominator of S is defined to be the object $o \in S$ that is most far away from q (i.e.,$o = \arg\max_{o \in S} dist(q, o)$).

(2) Let o be the distance dominator of S. A circle with q as the center and $d(q, o)$ as the radius is denoted by $C(q, r)$, where $r = d(q, o)$.

(3) Given a keyword k, for the objects containing k, the one who is closest to q is called the k-keyword nearest neighbor of q, denoted by $NN(q, k)$.

5.1. Pruning Strategies

The idea of TCoA1 algorithm is to find feasible solutions with the help of distance dominators. However, not every object can be distance dominators, so we will find the lower bound of distance to q for distance dominators by the following theorem.

Theorem 1. *Given a query $q = (\lambda, K, T)$, let FS be a feasible solution, let o be the distance dominator of FS, and let $S = \{NN(q, k_1), NN(q, k_2), ..., NN(q, k_m)\}$. Then, we have $d(q, o) \geq d_{LB}$, where $d_{LB} = \max_{o' \in S} dist(q, o')$.*

Proof. Assume $d(q, o) < d_{LB}$. Let o_f be the distance dominator of S, i.e., $d_{LB} = d(q, o_f)$, and o_f contains the keyword k_f. Since $d(q, o) < d_{LB}$, we get $o_f \notin FS$, and there must be an object $o'_f \in FS$ containing k_f, such that $d(q, o'_f) \leq d(q, o)$. Therefore, we have $d(q, o'_f) < d_{LB}$. This conclusion contradicts that o_f is $NN(q, k_f)$, so it is true that $d(q, o) \geq d_{LB}$. ■

If some distance dominators are too far from q, then the solutions obtained by such distance dominators would not be the optimal solution. Therefore, such distance dominators can be pruned by the following theorem.

Theorem 2. *Given a query $q = (\lambda, K, T)$, and the current optimal solution S. Let S' be any feasible solution, and let o be the distance dominator of S'. If $d(q, o) \geq d_{UB}$, where $d_{UB} = (1 - score_1(q, S))/\alpha$, then we have $score_1(q, S') \leq score_1(q, S)$.*

Proof. Assume when o is the distance dominator of S' and $d(q, o) \geq d_{UB}$, there is $score_1(q, S') > score_1(q, S)$. Since $score_1(q, S') = \alpha(1-d(q, o))+(1-\alpha) \min_{o_j \in S'} \Gamma_{o_j} \leq \alpha(1 - d(q, o)) + (1 - \alpha)$, we have $\alpha(1 - d(q, o)) + (1 - \alpha) > score_1(q, S)$, i.e., $d(q, o) < (1-score_1(q, S))/\alpha$, that is, $d(q, o) < d_{UB}$, which contradicts the hypothesis. ■

Fig. 3 shows the distance constraint when looking for distance dominators. Suppose o_1, o_2, o_3, o_4, o_5, and o_6 contain one query keyword in K respectively. Initially, we only need to consider the objects in the gray area (i.e., o_2, o_3, o_5) to be distance dominators. As the current optimal solution S changes, the upper bound d_{UB} will become smaller and smaller.

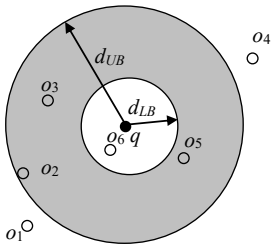


Fig. 3. Distance constraint for distance dominators

5.2. TCoA1 Algorithm Description

The steps of the algorithm are as follows:

Step 1: For each query keyword k_i in K, $NN(q, k_i)$ is found to form the current optimal solution S.

(1) For the m query keywords in K, the m TR-trees are found, respectively;

(2) For each TR$_{k_i}$-tree, a min heap $Minheap_i$ is created, with $d(q, elem)$ as the key, where $elem$ is the node or object of TR$_{k_i}$-tree;

(3) We create a max heap $maxHeap_i$ for each TR$_{k_i}$-tree, with Γ_{o_i} as the key, where o_i represents the object accessed in the TR$_{k_i}$-tree;

(4) For each TR$_{k_i}$-tree, $NN(q, k_i)$ is found with the best-first strategy [12] to form the current optimal solution S, i.e., $S = \{NN(q, k_1), NN(q, k_2)$,..., $NN(q, k_m)\}$.

The process of finding $NN(q, k_i)$ is as follows: The root node of TR$_{k_i}$-tree is kept in $Minheap_i$. Determine whether the top element of $Minheap_i$ is a node or an object. If it is a node, remove the node and store its children in $Minheap_i$. This process is repeated until the top element is an object, and the object is $NN(q, k_i)$.

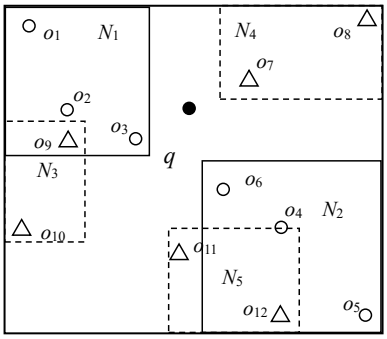
Example 2. Fig. 4 shows an example. Assume that the query keywords are "restaurant" and "gym". The initial state of $Minheap_1$ and $Minheap_2$ is shown in Fig. 4b. When S is found by using the best-first strategy for each TR-tree, the state of $Minheap_1$ and $Minheap_2$ is shown in Fig. 4c, we know that $NN(q, k_1) = o_3$, $NN(q, k_2) = o_7$, therefore, we have $S = \{o_3, o_7\}$.

Step 2: Using S, we can determine d_{LB}, which is used to identify the distance dominator. The object $o \in D$ will be put into the $maxHeap$ in ascending order $d(q, o)$. Once a distance dominator is put into the $maxHeap$, it will be used to find a feasible solution S'.

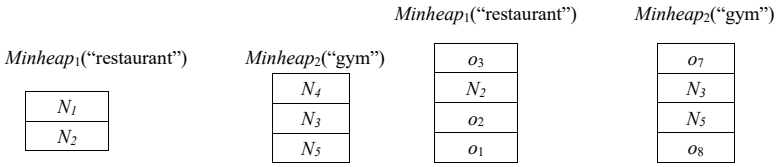
(1) Let $o_f = \arg\max_{o_f \in S} d(q, o_f)$, $d_{LB} = d(q, o_f)$;

(2) For each $o \in Minheap_i (1 \leq i \leq m)$, if $d(q, o) < d_{LB}$, then remove o from $Minheap_i$, and put it into $maxHeap_i$. All these objects will not be distance dominators according to Theorem 1.

(3) With best-first strategy, we look for a distance dominator $minobject$ from $Minheap_j$ $(1 \leq j \leq m)$, and put it into $maxHeap_j$ to find a new feasible solution S', which is formed by $minobject$ and the heads of all the max heaps except $maxHeap_j$. If $score_1(q, S) < score_1(q, S')$, S is replaced by S'.



(a) Object placement



(b) Initial state of $Minheap_1$ and $Minheap_2$

(c) The changed state of $Minheap_1$ and $Minheap_2$

Fig. 4. An example of step 1

(4) According to Theorem 2, if $d(q, minobject) < (1 - score_1(q, S))/\alpha$, then we repeat (3), otherwise the algorithm terminates.

In (3), once a distance dominator $minobject$ is found, all the objects in $C(q, d(q, minobject))$ have been put into the max heaps. But we need not combine all the objects in the max heaps with $minobject$ to find the feasible solution with the largest $score_1$. Instead, we only get the feasible solution S', which is formed by $minobject$ and the heads of all the max heaps except the max heap of $maxHeap_j$. The reason is that $minobject$ is the distance dominator of S', that is, $minobject$ is the object furthest away from q in S'. Since $score_1(q, S') = \alpha(1 - d(q, minobject)) + (1 - \alpha) \min_{o \in S'} \Gamma_o$, we know that $\alpha(1 - d(q, minobject))$ is a fixed value when we compute $score_1(q, S')$. Therefore, we create corresponding max heap $maxHeap_i$ for each TR_{k_i}-tree, with Γ_{o_i} as the key, and just take the head of all the max heaps except $maxHeap_j$ to form S', which has the largest $score_1$ among all the combinations.

Example 3. From Example 2, it is known that $d_{LB} = d(q, o_7)$. Assume that the current state of $Minheap_1$, $Minheap_2$, $maxHeap_1$, and $maxHeap_2$ is shown in Fig. 5a. Next, we first compare $d(q, o_6)$ with $d(q, o_9)$. From Fig. 4a, we know that $d(q, o_6) < d(q, o_9)$. Then, o_6 is removed from $Minheap_1$, and Γ_{o_6} is calculated and stored in $maxHeap_1$. The state of all the heaps is shown in Fig. 5b. Since $d(q, o_6) > d_{LB}$, o_6 will be the distance dominator of the next feasible solution S', we get $S' = \{o_6, o_7\}$.

The pseudo-code of TCoA1 algorithm is presented in Algorithm 1. The m min heaps are initialized, and $Minheap_i$ is used to store objects or nodes in the TR_{k_i}-tree (line 1).

Algorithm 1: TCoA1(D, q)

Input: A spatial database D, a query $q = (\lambda, K, T)$, where $K = \{k_1, ..., k_m\}$, $T = \{t_1, ..., t_m\}$
Output: The result S

Initialize m min heaps $Minheap_1, ..., Minheap_m$ with $d(q, elem)$ as key;
Initialize m max heaps $maxHeap_1, ..., maxHeap_m$ with Γ_{elem} as key;
$Lists = \emptyset$, $S = \emptyset$;
for *each query keyword* $k_i \in K$ **do**
    $Minheap_i \leftarrow$ the root in TR$_{k_i}$-tree;
    $Minheap_i =$ FindN($q, Minheap_i$, TR$_{ki}$-tree);
    $S \leftarrow Minheap_i.head$;
    $Lists \leftarrow Minheap_i$;
$d_{LB} = \max_{o \in S} d(q, o)$;
while $Lists \neq \emptyset$ **do**
    $Minheap_j = \arg \min_{Minheap_i \in Lists} d(q, Minheap_i.head)$;
    $minobject = Minheap_j.head$;
    Remove $Minheap_j.head$ from $Minheap_j$;
    $maxHeap_j \leftarrow minobject$;
    if $Minheap_j = \emptyset$ **then**
        Remove $Minheap_j$ from $Lists$;
    else
        $Minheap_j =$ FindN($q, Minheap_j$, TR$_{k_j}$-tree);
    if $d(q, minobject) < d_{LB}$ **then**
        continue;
    else
        if $d(q, minobject) < (1 - score_1(q, S))/\alpha$ **then**
            $S' = \emptyset$;
            for $(x = 1; x \leq m; x++)$ **do**
                if $x == j$ **then**
                    $S' \leftarrow minobject$;
                else
                    $S' \leftarrow maxHeap_x.head$;
            if $score_1(q, S') > score_1(q, S)$ **then**
                $S = S'$;
        else
            break;
return S;

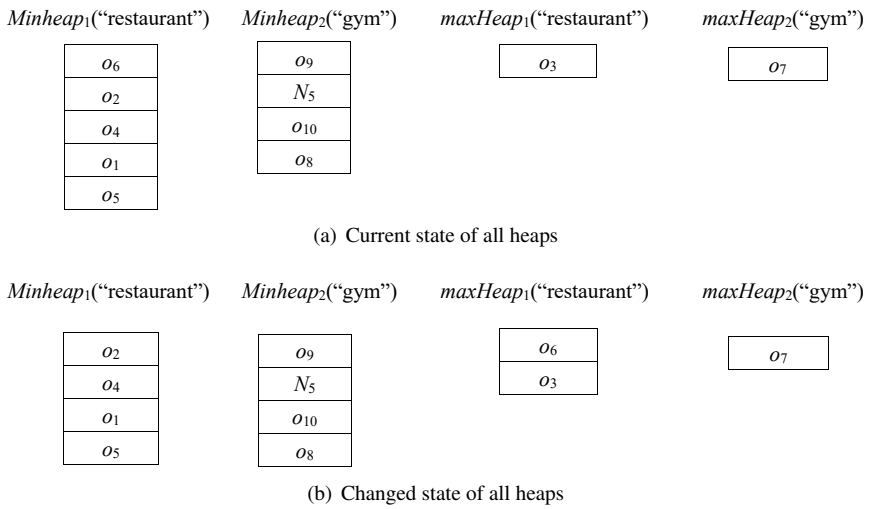


Fig. 5. An example of step 2

The m max heaps are initialized, and $maxHeap_i$ is used to store objects with keyword k_i (line 2). In lines 4-8, for each TR_{k_i}-tree corresponding to the query keyword k_i, Algorithm 2 is called to find $NN(q, k_i)$ (line 6). In line 7, $Minheap_i.head$ is $NN(q, k_i)$. After all the keyword nearest neighbors are found, the current optimal solution S is formed. In lines 10-32, for the heads of all the min heaps in $Lists$, the head $minobject$ with the smallest key is found and moved from $Minheap_j$ to $maxHeap_j$. After that, if $Minheap_j$ is not empty, the heap is processed until the head is an object. We process $minobject$ with d_{UB} and d_{LB}. In lines 22-30, for a distance dominator $minobject$, if $d_{LB} \leq d(q, minobject) < d_{UB} = (1 - score_1(q, S))/\alpha$, then a new feasible solution S' will be found. The current optimal solution will be updated by S' when $score_1(q, S') > score_1(q, S)$.

The pseudo-code of FindN algorithm for finding an object is presented in Algorithm 2. In lines 1-8, if the heap is not empty, then the heap will be traversed in the best-first strategy until the head of the heap is an object. In line 9, the changed heap is returned.

5.3. Time Complexity of TCoA1 Algorithm

We assume that the objects are uniformly distributed. According to [12], after finding the k nearest neighbors, the total cost is $O(klogk)$. In TCoA1 algorithm, there are m min heaps, which are used to find the nearest neighbor in the best-first strategy [12]. Each min heap is used separately. Each time only one of the min heap is chosen to find the nearest neighbor incrementally. And the found nearest neighbor will be put into the corresponding max heap. After the k nearest neighbors are put into a max heap, the total cost is $O(klogk)$. Once a distance dominator minobject from $Minheap_j$ is put into a max heap, a feasible solution is formed by minobject and the heads of all the max heaps except the max heap of $maxHeap_j$. The cost is $O(1)$. Let the maximum number of nearest neighbors found by min heap be k_a. So the time complexity of TCoA1 algorithm is $O(mk_alogk_a)$.

Algorithm 2: FindN(q, $heap$, TR$_{k_i}$-tree)

Input: a query q, a min heap $heap$ for keeping nodes or objects of TR$_{k_i}$-tree, TR$_{k_i}$-tree for the query keyword k_i
Output: $heap$

```
while heap ≠ ∅ do
    elem = heap.head;
    if elem is a node then
        Remove elem from heap;
        for each child e of elem do
            heap ← e;
    else
        break;
return heap;
```

6. TCoA2 Algorithm

We propose TCoA2 algorithm for the evaluation function $score_2(q, S)$ to solve the TCoSKQ problem.

The idea of TCoA2 algorithm is to find the central object o in ascending order of $d(q, o)$, which is used to find a feasible solution. We propose the following theorem to prune the central object that cannot form the optimal solution.

The current optimal solution obtained during the search process is defined as follows: If S is the current optimal solution found by o, then for any result set S_i found by o_i so far, we have $score(q, S_i, o_i) \leq score(q, S, o)$.

Theorem 3. *Given a query $q = (\lambda, K, T)$, the current optimal solution S found by o. Let S' be any feasible solution, whose best center object is o'. If $d(q, o') \geq (1 - score(q, S, o))/\beta$, we have $score_2(q, S') \leq score(q, S, o)$.*

Proof. Assume that if $d(q, o') \geq (1-score(q, S, o))/\beta$, we have $score_2(q, S') > score(q, S, o)$. Since o' is the best center object of S', we have $score_2(q, S') = \beta(1 - d(q, o')) + (1 - \beta)score_1(q', S') \leq \beta(1 - d(q, o')) + (1 - \beta)$, where $q' = (o'.\lambda, K, T)$. Thus, $\beta(1 - d(q, o')) + (1 - \beta) > score(q, S, o)$, that is, $d(q, o') < (1 - score(q, S, o))/\beta$, which contradicts the hypothesis. ■

It can be known from Theorem 3 that when the central object is searched from near to far, if the distance from the central object to the query reaches an upper bound, then it is not necessary to continue to search for the central object, and the search process can be terminated early.

The following example demonstrates that the center object found firstly may not be the best central object.

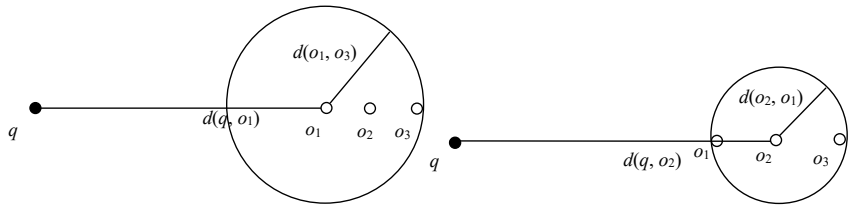
Example 4. Fig. 6 shows an example. Let $max\_dist = 100, dist(q, o_1) = 70, dist(o_1, o_3) = 20, dist(o_2, o_1) = 10, \alpha = 0.5, \beta = 0.3$, the feasible solution $S = \{o_1, o_2, o_3\}$, and $\min_{o_j \in S} \Gamma_{o_j}$=0.5.

(1) When S is found by the center object o_1, we have $score(q, S, o_1) = \beta(1 - d(q, o_1)) + (1 - \beta)score_1(q', S) = \beta(1 - d(q, o_1)) + (1 - \beta)(\alpha(1 - d(o_1, o_3)) + (1 - \alpha)\min_{o_j \in S} \Gamma_{o_j}) = 0.3 * (1 - 70/100) + 0.7 * (0.5 * (1 - 20/100) + 0.5 * 0.5) = 0.545$.

(2) When S is found by the center object o_2, we have $score(q, S, o_2) = \beta(1 - d(q, o_2)) + (1 - \beta)score_1(q', S) = \beta(1 - d(q, o_2)) + (1 - \beta)(\alpha(1 - d(o_2, o_1)) + (1 - \alpha)\min_{o_j \in S} \Gamma_{o_j}) = 0.3 * (1 - 80/100) + 0.7 * (0.5 * (1 - 10/100) + 0.5 * 0.5) = 0.55$.

(3) When S is found by the center object o_3, we have $score(q, S, o_3) = \beta(1 - d(q, o_3)) + (1 - \beta)score_1(q', S) = \beta(1 - d(q, o_3)) + (1 - \beta)(\alpha(1 - d(o_3, o_1)) + (1 - \alpha)\min_{o_j \in S} \Gamma_{o_j}) = 0.3 * (1 - 90/100) + 0.7 * (0.5 * (1 - 20/100) + 0.5 * 0.5) = 0.485$.

According to the above result, it can be seen that only o_2 is the best central object of S.



(a) Find S by o_1 (b) Find S by o_2

Fig. 6. Find the best central object for S

Example 5. Consider Example 4 again. Assume that the current best solution S' is found by o' and $score(q, S', o') = 0.775$. By Theorem 3, we have $(1 - score(q, S', o'))/\beta = 0.75$. Since $d(q, o_1) = 0.7 < 0.75$, we should compute $score(q, S, o_1)$, which is 0.545. Because $0.545 < 0.775$, the current optimal solution is unchanged. Since $d(q, o_2) = 0.8 > 0.75$, the query is terminated according to Theorem 3. It is no longer necessary to calculate $score(q, S, o_2)$.

The main idea of TCoA2 algorithm is shown in Fig. 7.

The pseudo-code of TCoA2 algorithm is presented in Algorithm 3. The m min heaps are initialized, and $heap_i$ is used to store objects or nodes in the TR_{k_i}-tree (line 1). For each TR_{k_i}-tree, we store its root node in the corresponding heap (lines 3-5). In lines 6-23, for the head of each heap in $Lists$, the element $elem$ with the smallest key is found and removed. If $elem$ is a node, all children of $elem$ are stored in the corresponding heap. If $elem$ is an object, then it is a central object, which is used to find S'. In lines 16-21, if $d(q, elem) < (1 - Cscore))/\beta$, then it is possible to find the optimal solution by $elem$ according to Theorem 3. TCoA1* algorithm is called to find a feasible solution, and the final solution is updated.

TCoA1* algorithm is got by modifying TCoA1 algorithm as follows:

a. Different input. TCoA1* algorithm needs to know which object is to be the center object to find the result set;

Algorithm 3: TCoA2(D, q)

Input: A spatial database D, a query $q = (\lambda, K, T)$, where $K = \{k_1, ..., k_m\}$, $T = \{t_1, ..., t_m\}$

Output: The result S

Initialize m min heaps $heap_1, ..., heap_m$ with $d(q, elem)$ as key;
$Lists = \emptyset$, $S = \emptyset$, $Cscore = 0$;
for *each query keyword* $k_i \in K$ **do**
    $heap_i \leftarrow$ the root in TR$_{k_i}$-tree;
    $Lists \leftarrow heap_i$;
while $Lists \neq \emptyset$ **do**
    $heap_j = \arg \min_{heap_i \in Lists} d(q, heap_i.head)$;
    $elem = heap_j.head$;
    Remove $heap_j.head$ from $heap_j$;
    if $elem$ *is a node* **then**
        for *each child* e *of* $elem$ **do**
            $heap_j \leftarrow e$;
    else
        if $heap_j = \emptyset$ **then**
            Remove $heap_j$ from $Lists$;
        if $d(q, elem) < (1 - Cscore)/\beta$ **then**
            $q' = (elem.\lambda, K, T)$;
            $S' =$ TCoA1*$(D, q', elem)$;
            if $score(q, S', elem) > Cscore$ **then**
                $S = S'$;
                $Cscore = score(q, S', elem)$;
        else
            break;
return S;

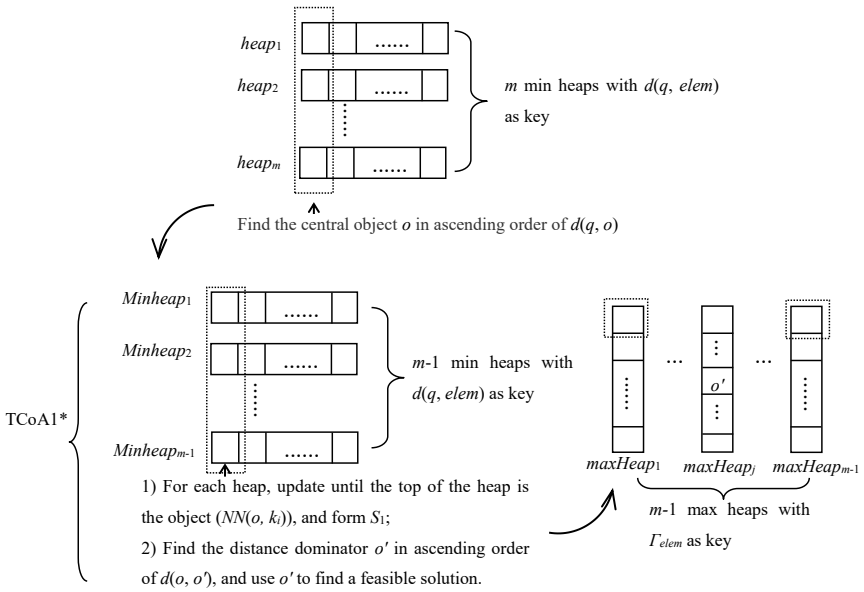


Fig. 7. The main idea of TCoA2 algorithm

b. For the input central object o, TCoA1* algorithm does not need to create a min heap and a max heap for the TR-tree corresponding to the keyword contained in o;

c. All result sets found by TCoA1* algorithm must contain the input central object.

To support the correctness of TCoA1* algorithm, we give the following Theorems by rewriting Theorem 1 and 2, respectively.

Theorem 4. *Given q'=$(o_i.\lambda, q.K, q.T)$, where o_i is a central object, let FS be a feasible solution containing o_i, let o be the distance dominator of FS, and let $S = \{NN(q', k_1)$,..., o_i, ..., $NN(q', k_m)\}$. Then, we have $d(q', o) \geq d_{LB}$, where $d_{LB} = \max_{o' \in S} d(q', o')$.*

Proof. The proof is similar to that of Theorem 1. ■

Theorem 5. *Given q'=$(o_i.\lambda, q.K, q.T)$, where o_i is a central object, and the current optimal solution S containing o_i. Let S' be any feasible solution containing o_i, and let o be the distance dominator of S'. If $d(q', o) \geq d_{UB}$, where $d_{UB} = (1 - score_1(q', S))/\alpha$, then we have $score_1(q', S') \leq score_1(q', S)$.*

Proof. The proof is similar to that of Theorem 2. ■

For each found central object o, TCoA1* algorithm is called to find feasible solution S' containing o, where S' is the best solution with o being a central object according to Theorem 4 and 5.

Correctness analysis of TCoA2 algorithm: According to Definition 1, for any feasible solution S, it is necessary to calculate $score(q, S, o_i)$, where o_i is a best central object. But for S, we could not predict which object in S is the best central object, so we have to try each object o as a central object in ascending order of $d(q, o)$. For

TCoA2 algorithm, assume that the current optimal solution S_p is found by o_p, and the best central object of any feasible solution S is o. According to Theorem 3, if $d(q,o) \geq (1 - score(q, S_p, o_p))/\beta$, then we have $score_2(q, S) \leq score(q, S_p, o_p)$. Therefore, TCoA2 algorithm can return the correct result.

Time complexity of TCoA2 algorithm: We assume that the objects are uniformly distributed. In TCoA2 algorithm, there are m min heaps, which are used to find the nearest neighbor in the best-first strategy [12]. Once a nearest neighbor is found, it is used to call TCoA1* algorithm, whose time complexity is the same with that of TCoA1 algorithm. Let the maximum number of nearest neighbors found by min heap be k_b, and let the maximum number of nearest neighbors found by min heap in TCoA1* algorithm be k_a. So the time complexity of TCoA2 algorithm is $O(mk_blogk_b + mk_bmk_alogk_a)$.

7. Experiment and Result Analysis

All the algorithms are implemented in Java 1.7.0. All the experiments have been performed on a Windows 7 PC with an Intel(R) Core(TM) i5-2450M CPU and 4G RAM.

We use the datasets Oldenburg (https://www.cs.utah.edu /∼lifeifei/ SpatialDataset.htm) and GN (extracted from the Geographic Names Information System in USA, https://www.usgs.gov/), and generate randomly keyword information and valid time information for each object in the datasets. Oldenburg contains 6,105 objects and 59 different keywords. GN contains 2288631 objects and 1865 different keywords. We assign one keyword to each object using the Zipf distribution [13] in both datasets. The starting time stamp of objects in both datasets is set to follow the Gaussian distribution, and the ending time stamp of objects is computed by the starting time stamp plus the length of time interval which is in the range of [6, 12]. Each query location is randomly read from the dataset, and the query keywords and query time interval are randomly assigned to the query. We randomly produce 100 queries and report the average results.

7.1. TCoA1 algorithm and Baseline algorithm

The max heaps are used to obtain a feasible solution in step 2 of TCoA1 algorithm. In order to test the performance of those max heaps, we propose a baseline algorithm (Baseline) for comparison.

Baseline algorithm is got by modifying TCoA1 algorithm as follows:

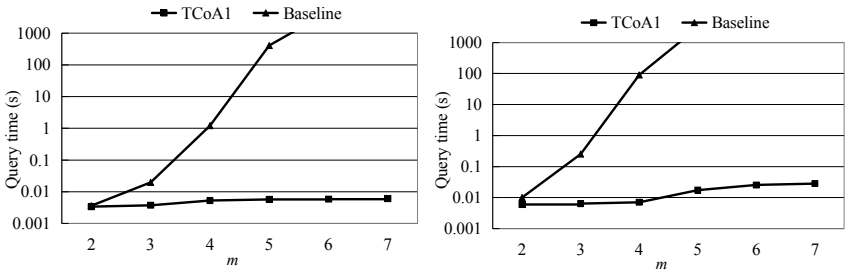
a. In step 1 (3) in section 5.2, the max heaps are replaced by lists. For each TR_{k_i}-tree, the list $list_i$ is created, which keeps objects that have been visited in the TR_{k_i}-tree.

b. In step 2 (3) in section 5.2, after finding the distance dominator $minobject$, we combine $minobject$ with all objects in each list except the list containing $minobject$ to get multiple feasible solutions. In each feasible solution, one object is taken from each list. We retain the solution S' with the largest $score_1$. Finally, the final result is updated with S'.

We should note that the pruning strategies for the distance dominators in TCoA1 algorithm are applicable in Baseline algorithm. Therefore, Baseline algorithm uses the pruning strategies too.

(1) Effect of the number of query keywords m

Fig. 8 shows the influence of m on the query time of TCoA1 algorithm and Baseline algorithm. We set $\alpha = 0.6$. According to Fig. 8, TCoA1 algorithm is faster than Baseline algorithm. It is because after finding the distance dominator $minobject$, TCoA1 algorithm directly obtains head of each heap except the heap containing $minobject$, and then combines with $minobject$ to get a feasible solution. However, Baseline algorithm needs to obtain all the objects in each list except the list containing $minobject$, and to combine with $minobject$ to get a large number of feasible solutions. Finally, we retain the feasible solution with the largest $score_1(q, S')$ to update the final result. The query time of both algorithms increases when m increases. This is because the total number of objects containing query keywords may increase when m increases. This may result in an increase in the total number of the distance dominators used. It can also be seen from Fig. 8 that TCoA1 algorithm are scalable.



(a) Query time (Oldenburg) (b) Query time (GN)

Fig. 8. Query time versus m

(2) Effect of α

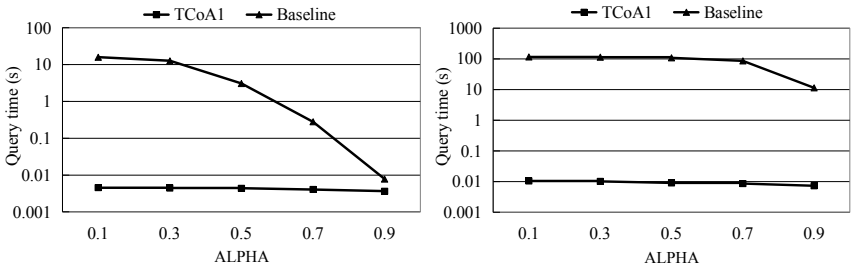
The α is a parameter used to adjust the distance and time. When α changes from 1 to 0, more weight is assigned to temporal relevance. When α is close to 1, TCoSKQ is more related to positional relevance.

Fig. 9 shows the influence of α on the query time of TCoA1 algorithm and Baseline algorithm. We set $m = 4$. According to Fig. 9, with α increasing, the query time of TCoA1 algorithm and Baseline algorithm decreases. This is because when α increases, the smaller $\max_{o \in S} d(q, o)$ is, the larger the $score_1(q, S)$ is. TCoSKQ prefer finding the result close to the query location. It is easier to find the result that satisfies the condition in a small range, so that few distance dominators are used.

7.2. TCoA1 algorithm and TCoA1NoPrune algorithm

In order to test the pruning effect on the total number of the distance dominators used (N_{du}), we remove a pruning strategy of TCoA1 algorithm (i.e., Theorem 2), then get TCoA1NoPrune algorithm. Next, we compare TCoA1 algorithm and TCoA1NoPrune algorithm on N_{du} and query time using the two datasets.

(1) Effect of m

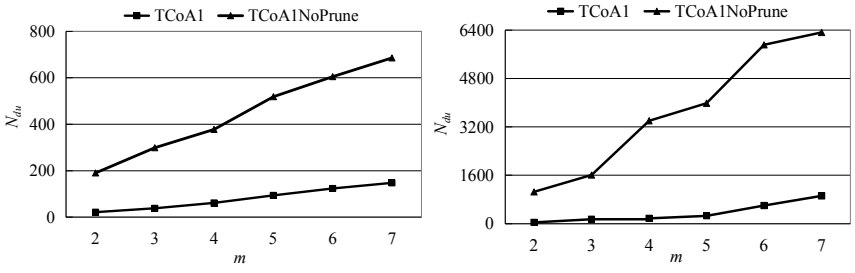


(a) Query time (Oldenburg) (b) Query time (GN)

Fig. 9. Query time versus α

Fig. 10 shows the influence of m on N_{du} in the query. We set $\alpha = 0.6$. According to Fig. 10, TCoA1 algorithm uses less total number of distance dominators than TCoA1NoPrune algorithm does. This is because TCoA1 algorithm uses pruning strategy to prune a large number of distance dominators. N_{du} in TCoA1NoPrune algorithm is the total number of objects containing the query keywords and the distance to q is not less than d_{LB}. With m increasing, N_{du} in TCoA1 algorithm and TCoA1NoPrune algorithm increases. This is because the total number of objects containing query keywords may increase when m increases.

Fig. 11 shows the influence of m on query time. We set $\alpha = 0.6$. According to Fig. 11, TCoA1 algorithm is faster than TCoA1NoPrune algorithm. As m increases, the query time of TCoA1 algorithm and TCoA1NoPrune algorithm increases. This is because the query time of the algorithms is related to N_{du}.

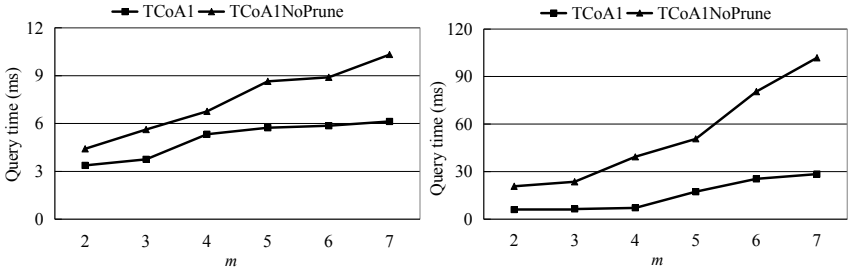


(a) N_{du} (Oldenburg) (b) N_{du} (GN)

Fig. 10. N_{du} versus m

(2) Effect of α

Fig. 12 shows the influence of α on N_{du} in the query. We set $m = 4$. According to Fig. 12, α has no impact on N_{du} in TCoA1NoPrune algorithm. This is because the distance dominator is not pruned in TCoA1NoPrune algorithm, and N_{du} is the total number of objects containing the query keywords and the distance to q is not less than d_{LB}.

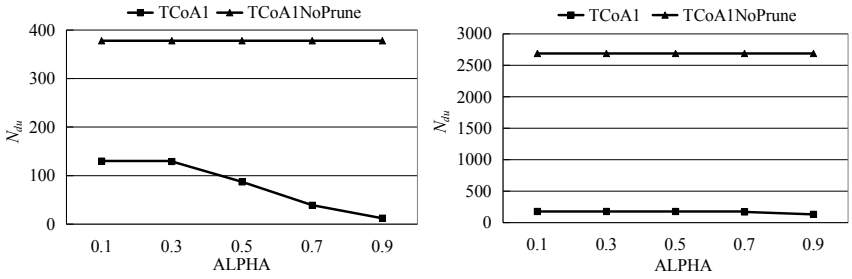


(a) Query time (Oldenburg) (b) Query time (GN)

Fig. 11. Query time versus m

As α increases, N_{du} in TCoA1 algorithm decreases. This is because when α increases, TCoSKQ prefer positional relevance. It is easier to find the result that satisfies the condition in a small range, so that few distance dominators are used.

Fig. 13 shows the influence of α on query time. We set $m = 4$. As shown in Fig. 13, α has no impact on the query time of TCoA1NoPrune algorithm, and the query time in TCoA1 algorithm decreases with α increasing. This is because the query time of the algorithms is related to N_{du}.



(a) N_{du} (Oldenburg) (b) N_{du} (GN)

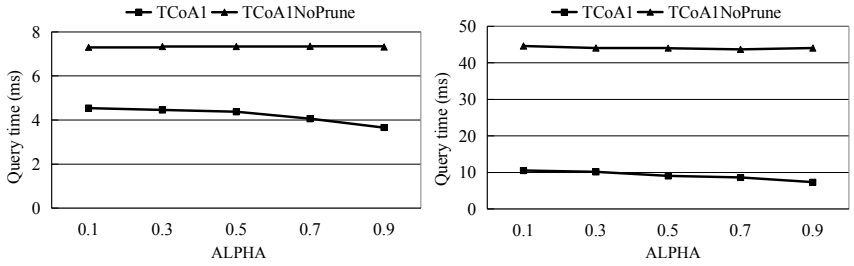
Fig. 12. N_{du} versus α

7.3. TCoA2 algorithm and TCoA2NoPrune algorithm

In order to test the pruning effect on the total number of center objects used (N_{co}), we remove the pruning strategy of TCoA2 algorithm (i.e., Theorem 3), then get TCoA2NoPrune algorithm. Next, we compare TCoA2 algorithm and TCoA2NoPrune algorithm on N_{co} and query time using the two datasets.

(1) Effect of m

Fig. 14 shows the influence of m on N_{co} in the query. We set $\alpha = 0.6$ and $\beta = 0.6$. According to Fig. 14, N_{co} in TCoA2 algorithm is less than TCoA2NoPrune algorithm.

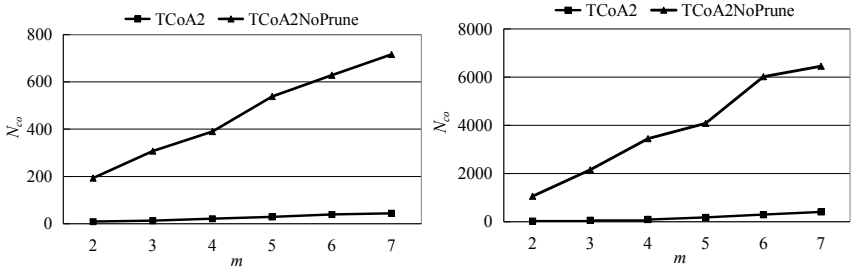


(a) Query time (Oldenburg) (b) Query time (GN)

Fig. 13. Query time versus α

This is because TCoA2 algorithm uses pruning strategy to prune a large number of center objects. N_{co} in TCoA2NoPrune algorithm is the total number of objects containing the query keywords. When m increases, N_{co} in TCoA2 algorithm and TCoA2NoPrune algorithm increases. This is because the total number of objects containing query keywords may increase when m increases.

Fig. 15 shows the influence of m on query time. We set $\alpha = 0.6$ and $\beta = 0.6$. With m increasing, the query time of TCoA2 algorithm and TCoA2NoPrune algorithm increases, and TCoA2 algorithm is faster than TCoA2NoPrune algorithm. This is because the query time of the algorithms is related to N_{co}.



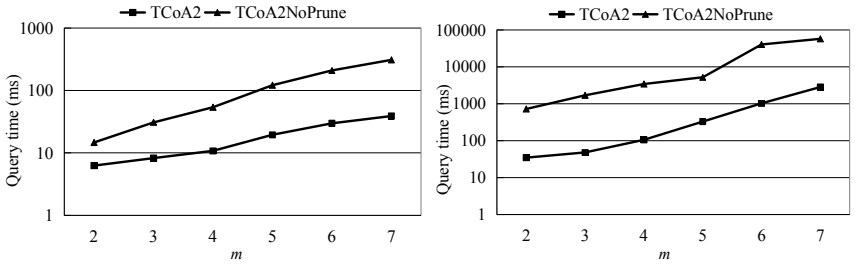
(a) N_{co} (Oldenburg) (b) N_{co} (GN)

Fig. 14. N_{co} versus m

(2) Effect of β

The β is a parameter used to adjust the distance and $score_1$. When β changes from 1 to 0, more weight is assigned to $score_1$. When β is close to 1, TCoSKQ is more related to positional relevance.

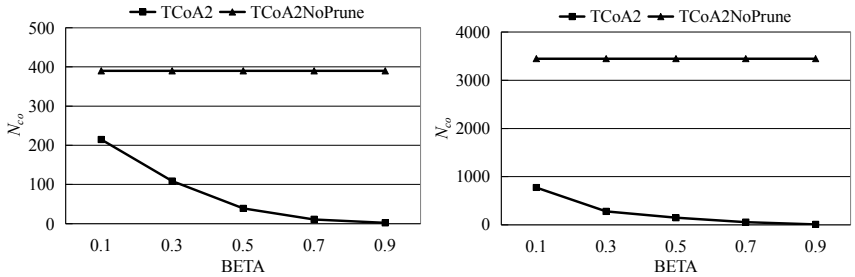
Fig. 16 shows the influence of β on N_{co} in the query. We set $m = 4$ and $\alpha = 0.6$. According to Fig. 16, β has no impact on N_{co} in TCoA2NoPrune algorithm. This is because there is no pruning strategy in TCoA2NoPrune algorithm, and N_{co} is the total number of objects containing the query keywords. As β increases, N_{co} in TCoA2 algo-



(a) Query time (Oldenburg) (b) Query time (GN)

Fig. 15. Query time versus m

rithm decreases. This is because when β increases, TCoSKQ prefer positional relevance. It is easier to find the result that satisfies the condition in a small range, so that few center objects are used.



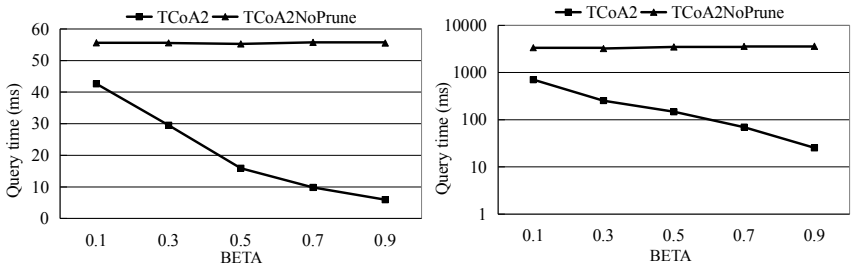
(a) N_{co} (Oldenburg) (b) N_{co} (GN)

Fig. 16. N_{co} versus β

Fig. 17 shows the influence of β on query time. We set $m = 4$ and $\alpha = 0.6$. As shown in Fig. 17, β has no impact on the query time of TCoA2NoPrune algorithm, and the query time in TCoA2 algorithm decreases with β increasing. This is because the query time of the algorithms is related to N_{co}.

8. Conclusions

This paper presents a new query, time-aware collective spatial keyword query (TCoSKQ). For different needs of users, we define two new evaluation functions, $score_1$ and $score_2$, and adopt the TR-tree index structure. For the evaluation function $score_1$, we propose pruning strategies for effectively pruning the number of the distance dominators used, and give TCoA1 algorithm for solving the query problem. For the evaluation function $score_2$, we propose effective pruning strategies to prune the number of the center objects used,



(a) Query time (Oldenburg) (b) Query time (GN)

Fig. 17. Query time versus β

and give TCoA2 algorithm. Finally, the efficiency and scalability of the two algorithms are verified. In the future work, other evaluation functions could be proposed.

Acknowledgments. This work was supported by the Key Research and Development Program of Hebei Province, China (No. 18270307D).

References

1. Cao, X., Cong, G., Jensen, C.S., Ooi, B.C.: Collective spatial keyword querying. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. pp. 373–384 (2011)
2. Cary, A., Wolfson, O., Rishe, N.: Efficient and scalable method for processing top-k spatial boolean queries. In: Proceedings of the 22nd International Conference on Scientific and Statistical Database Management, SSDBM 2010, Heidelberg, Germany. pp. 87–95 (2010)
3. Chan, H.K.H., Long, C., Wong, R.C.W.: Inherent-cost aware collective spatial keyword queries. In: Proceedings of the 15th International Symposium on Advances in Spatial and Temporal Databases, SSTD 2017, Arlington, VA, USA. pp. 357–375 (2017)
4. Chan, H.K.H., Long, C., Wong, R.C.W.: On generalizing collective spatial keyword queries. IEEE Transactions on Knowledge and Data Engineering 30(9), 1712–1726 (2018)
5. Chen, G., Zhao, J., Gao, Y., Chen, L., Chen, R.: Time-aware boolean spatial keyword queries. IEEE Transactions on Knowledge and Data Engineering 29(11), 2601–2614 (2017)
6. Chen, Z., Zhao, T., Liu, W.: Time-aware spatial keyword cover query. Data & Knowledge Engineering 122, 81–100 (2019)
7. Chen, Z., Zhou, T., Liu, W.: Direction aware collective spatial keyword query. Journal of Chinese Computer Systems 35(5), 999–1004 (2014)
8. Felipe, I.D., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, Cancún, Mexico. pp. 656–665 (2008)
9. Gao, Y., Zhao, J., Zheng, B., Chen, G.: Efficient collective spatial keyword query processing on road networks. IEEE Transactions on Intelligent Transportation Systems 17(2), 469–480 (2016)
10. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, USA. pp. 47–57 (1984)

11. He, P., Xu, H., Zhao, X., Shen, Z.: Scalable collective spatial keyword query. In: 31st IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2015, Seoul, South Korea. pp. 182–189. IEEE (2015)
12. Hjaltason, G.R., Samet, H.: Distance browsing in spatial databases. ACM Transactions on Database Systems 24(2), 265–318 (1999)
13. Joachims, T.: A statistical learning model of text classification for support vector machines. In: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2001, New Orleans, Louisiana, USA. pp. 128–136 (2001)
14. Li, Z., Lee, K.C., Zheng, B., Lee, W.C., Lee, D., Wang, X.: Ir-tree: An efficient index for geographic document search. IEEE Transactions on Knowledge and Data Engineering 23(4), 585–599 (2011)
15. Liu, W., Fu, Y., Chen, Z.: New collective query processing method based on spatial keyword. Journal of Chinese Computer Systems 34(8), 1831–1836 (2013)
16. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA. pp. 71–79 (1995)
17. Su, S., Zhao, S., Cheng, X., Bi, R., Wang, J.: Group-based collective keyword querying in road networks. Information Processing Letters 118, 83–90 (2017)
18. Tao, Y., Sheng, C.: Fast nearest neighbor search with keywords. IEEE transactions on knowledge and data engineering 26(4), 878–888 (2014)
19. Vaid, S., Jones, C.B., Joho, H., Sanderson, M.: Spatio-textual indexing for geographical search on the web. In: Proceedings of the 9th International Symposium on Advances in Spatial and Temporal Databases, SSTD 2005, Angra dos Reis, Brazil. pp. 218–235 (2005)
20. Zhang, P., Lin, H., Yao, B., Lu, D.: Level-aware collective spatial keyword queries. Information Sciences 378, 194–214 (2017)
21. Zhao, J., Gao, Y., Chen, G., Chen, R.: Towards efficient framework for time-aware spatial keyword queries on road networks. ACM Transactions on Information Systems 36(3), 24:1–24:48 (2018)
22. Zhao, S., Cheng, X., Su, S., Shuang, K.: Popularity-aware collective keyword queries in road networks. GeoInformatica 21(3), 485–518 (2017)

Zijun Chen received the bachelor's degree from the Northeast Heavy Machinery Institute, China, the master's degree from Yanshan University, and the PhD degree from Fudan University in 2002, all in computer science. Since 1995, he has been with the School of Information Science and Engineering, Yanshan University, Qinhuangdao, China, where he is currently a professor. His research interests include moving object databases and spatio-temporal databases.

Tingting Zhao received the bachelor's degree in computer science and technology from North China University of Science and Technology, China, in 2016. She received the master's degree in the School of Information Science and Engineering, Yanshan University, China, in 2019. Her research interest includes spatio-temporal databases.

Wenyuan Liu received the bachelor's and master's degrees from the Northeast Heavy Machinery Institute, China, and the PhD degree from the Harbin Institute of Technology in 2000, all in computer science. Since 1996, he has been with the School of Information Science and Engineering, Yanshan University, Qinhuangdao, China, where he is currently

a professor. His research interests include spatio-temporal databases, network sensing and mobile networks.

Received: January 31, 2020; Accepted: May 15, 2021.