IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 27 August 2022, accepted 6 October 2022, date of publication 10 October 2022, date of current version 14 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3213261

RESEARCH ARTICLE

Why Questions on Time-Aware Spatial Keyword Queries in Traffic Networks

XIAO JIA[1], YANHONG LI[2], YIHUA LAN[1], JINLIANG HUANG[2], AND XINGANG ZHANG[1]

[1]Henan Engineering Research Center of Intelligent Processing for Big Data of Digital Image, School of Computer Science and Technology, Nanyang Normal University, Nanyang 473061, China

[2]College of Computer Science, South-Central Minzu University, Wuhan 430074, China

Corresponding author: Yanhong Li (liyanhong@mail.scuec.edu.cn)

This work was supported by the Key Scientific Research Projects of Colleges and Universities in Henan Province of China under Grant 20A520031, Grant 21A520032, and Grant 21A520033.

ABSTRACT Time-aware spatial keyword queries in traffic networks (*TSKQT*) aim to retrieve top-*k* result objects based on the ranking score considering spatial proximity, textual relevance, and temporal similarity simultaneously. However, due to inappropriate query parameter settings, the query result set may contain one or more objects not expected by the user. To improve the usability of query results, we propose and study the why question on time-aware spatial keyword queries in the traffic network (*WhyTSKQT*) for the first time. Specifically, a hybrid index structure, *TTG-tree*, is proposed to effectively organize the traffic network information and the positional, textual, and temporal information of objects. Moreover, several pruning strategies are presented to filter out massive objects irrelevant to the query. By analyzing the keywords contained by original result objects and studying the relationship between the time intervals of why objects and the query time interval, we can reasonably construct the candidate query keyword set and query time set to form the candidate refined queries, so as to minimize the refined queries to be evaluated. In addition, several optimization techniques are proposed to further speed up the acquisition of the lowest-cost refined query. Finally, extensive experiments are carried out on two traffic networks to verify the efficiency of the proposed method.

INDEX TERMS Time-aware spatial keyword query, why question, query refinement, traffic network.

I. INTRODUCTION

With the increasing popularity of location-based services and continuous improvement of location-related technologies, spatial keyword query, as the core technology of location-based services, has become a research hotspot. Due to the continuous progress of information collection technology, the location-textual data in web space is becoming increasingly rich and diverse. For example, an object in web space contains not only location-textual information, but also temporal information, direction and other numerical attribute information, which undoubtedly makes spatial keyword queries more personalized. The time-aware spatial keyword query is one of the representatives of personalized queries. It considers not only the location and textual information of queries and objects but also their temporal information.

In practical applications, the query user and the object are located on the road of the traffic network, and the distance between them depends on the traffic network structure, that is, the length of the shortest path between them. Therefore, we focus on the Time-aware Spatial Keyword Query in Traffic networks (*TSKQT*), which aims to retrieve top-k objects based on a ranking function taking into account spatial proximity, textual relevance, and temporal similarity simultaneously.

Figure 1 illustrates an example of *TSKQT*, and 12 objects are located on the roads of the traffic network. The textual and temporal information of objects is shown in Table 1. The temporal information of an object o is its effective time interval in the form of [$o.starttime$,$o.endtime$]. Suppose the user issues a query q with the keywords ''bar'' and ''pop'',

The associate editor coordinating the review of this manuscript and approving it for publication was Congduan Li.

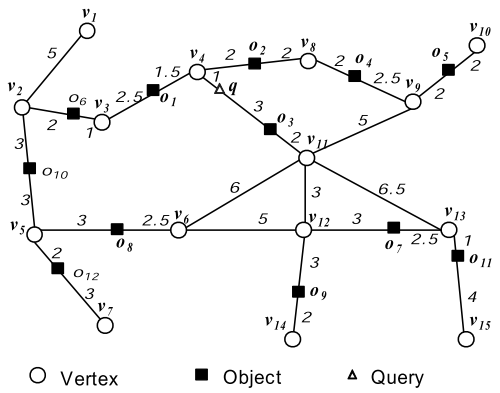


FIGURE 1. An example of the *TSKQT* query.

TABLE 1. Information about objects in Figure 1.

Obj	Textual and temporal information
o_1	$\{bar, pop\}$,[11:00:17:00]
o_2	$\{bar, pop, rock\}$,[10:00:16:00]
o_3	$\{bar, rock, pub\}$,[18:00:24:00]
o_4	$\{bar, rock, samba\}$,[06:00:12:00]
o_5	$\{bar, samba\}$,[17:00:24:00]
o_6	$\{bar, pub\}$,[20:00:05:00]
o_7	$\{pub, samba\}$,[10:00:18:00]
o_8	$\{pop, pub\}$,[08:00:12:00]
o_9	$\{rock, pub, samba\}$,[11:00:17:00]
o_{10}	$\{pop, rock\}$,[16:00:23:00]
o_{11}	$\{pop, rock\}$,[13:30:21:00]
o_{12}	$\{bar, samba\}$,[13:30:18:00]

and the query time is 10:00-13:00. Firstly, o_7 and o_9 are pruned because they do not contain any query keyword, and o_3, o_5, o_6, o_{10}, o_{11} and o_{12} are ignored since they violate the temporal constraints. The remaining objects o_1, o_2, o_4, and o_8, which contain some query keywords and satisfy the temporal requirements, are regraded as candidate objects. Then, by calculating the comprehensive similarity based on the network distance, textual relevance and temporal similarity between candidate objects and the query, the current top-3 result objects (o_1, o_2, and o_4) are obtained.

Unfortunately, after issuing a TSKQT query, the user may find that the query result set contains one or more data objects he does not expect, which will confuse him and cause him to question the query result. Continuing with the above example, assume that in the current Top-3 result set, o_4 is an unexpected object, called a why object. The user may wonder why the result set contains object o_4? How to adjust query settings to make o_4 disappear from the result set while preserving the non-why result objects as much as possible? The above problems are called why questions. If the query system can reasonably explain the why question raised by the user, and provide he with a refined query that can be obtained only by slightly modifying the original query, so as to exclude the why-object from the query result, it will greatly improve the usability of query results and the satisfaction of users with query results.

At present, there are relatively few studies on the why question. Gao et al. [1] studied the why-not and why questions in reverse Top-k queries. Their solution to why questions adopts the same method as the why-not question, that is, query refinement. However, so far, there are no relevant research results to solve the why question of spatial keyword queries. Considering that in the practical application, 1) objects are usually located on the roads the traffic network and their mutual distance depends on the specific structure of the underlying traffic network; 2) in addition to some keywords, the object description also contains some temporal information, we propose and study the why question of TSKQT queries in traffic networks for the first time.

Since *Query Refinement* is suitable for solving the why-not question and has achieved good processing results, this paper also uses *Query Refinement* to answer why questions on TSKQT queries (WhyTSKQT). An intuitive way to deal with this issue is to keep the original query keywords and query time interval and shrink the value of k until all why objects are absent from the result set, thus obtaining the so-called basic refined query q_b. However, this may greatly reduce the number of returned results and/or make query modification expensive. If the refined query is obtained by modifying query keywords and query time to exclude why objects, too many keyword sets and query time intervals can be selected to form large amounts of candidate refined queries, which involves massive useless similarity calculations. To efficiently answer the WhyTSKQT query, there are two main challenges to be overcome. 1) How to reasonably organize the comprehensive information of the traffic network and its objects to prune great amounts of of unpromising objects and refined queries, thus speeding up the TSKQT query processing; 2) How to reasonably construct the query keyword set and query time set to form the candidate refinement queries so that the number of candidate refinement queries need to be checked will be as few as possible.

To meet the first challenge, a new hybrid index called *TTG-tree* is designed to organize the location, textual, and temporal information of objects in an efficient way. Firstly, the G-tree index is used to divide the traffic network, and the distance measurement of each traffic network area (sub graph) is retained, so as to facilitate the network distance calculation. Secondly, for each subgraph, a textual and temporal component that reasonably organizes the textual and temporal information of its objects is constructed. Thirdly, according to the occurrence frequency, keywords are divided into frequent and infrequent keywords, which are indexed in different ways to facilitate textual information processing. Moreover, four lemmas are proposed to reduce massive unqualified objects to further accelerate query processing.

Another important issue to be addressed is how to obtain possible query keyword sets and query time intervals to form candidate refined queries. Considering that adding keywords irrelevant to the original query result set may exclude

non-why objects in the result set and add more new result objects, we only consider the keywords existing in the original result objects. In particular, we sequentially extract new keywords from original result objects and add them to the original keyword set, and remove old keywords of why objects from the keyword set to form candidate keyword sets. In addition, in order to obtain the candidate query time intervals, so as to minimize the overlap rate between the time interval of why objects and the query time on the premise of minimizing the cost of query time modification, we analyze the relationship between the query time and the time intervals of why objects, and discuss it in four cases. The following are the main contributions of this paper:

- This paper identifies and formulates the why question on time-aware spatial keyword queries in the traffic network (WhyTSKQT). As far as we know, this is the first time to study the why question on such queries.
- A new hybrid index named *TTG-tree* is designed. *TTG-tree* mainly includes three components, the G-tree for the traffic network, the keyword list & the Bloom buffer for distinguishing, and the textual & temporal part for each node of the G-tree. Moreover, four lemmas are presented to prune massive objects unrelated to WhyTSKQT queries.
- An efficient TTG-tree-based query processing method is presented, and several optimization techniques are also proposed to further improve the performance of this method. In addition, we study the possibility of extending this scheme to deal with the why question on regional queries and ordinary SKQ queries.
- Extensive experiments are carried out on two real traffic networks, and the experiment result shows that the proposed methods are efficient and scalable.

The rest of this paper is organized as follows. Section II reviews the relevant work, and Section III introduces the definition and preliminaries of the problem. In Section IV, we detail the hybrid index *TTG-tree*, and then Section V presents the *TTG-tree* based solution and three optimization techniques. Section VI studies the possibility of extending our method to handle the why question on other types of queries. Section VII introduces the experimental study. Finally, the conclusion is given in Section VIII.

II. RELATED WORK

A. SPATIAL KEYWORD QUERIES

In recent years, the spatial keyword query has attracted widespread attention and become a research hotspot, and many research results have been proposed. Felipe et al. [2] constructed IR^2 tree to solve such queries in Euclidean space. Huang et al. [3] studied the processing of continuous spatial keyword queries. In addition, several well-known indexing technologies, such as S^2I, I^3, and IL quadtree [4], have been proposed. Variants of spatial keyword query have also been studied, such as reverse spatial-textual kNN search, best keyword coverage search [5], Clue-based spatio-textual queries [6], semantic-aware spatial keyword queries [7], interactive spatial textual queries [8], location-aware fault-tolerant keyword search [9], collective spatial keyword queries of moving objects [10], etc.

Carlsson et al. studied spatial keyword query processing in the traffic network and designed the RNI index structure. Then, a composite indexing structure was designed [11], including the road network R-tree, the adjacent B-tree, the mapping B-tree, and inverted files. Keywords and road network distance information can be considered in query processing, which is conducive to improving processing efficiency. Guo et al. [12] discussed the continuous kNN search of spatial keywords on the traffic network, and realized the continuous monitoring of query result objects by establishing a safe section on the traffic network. Zhang et al. [13] discussed differentiated spatial keyword queries in the traffic network. Moreover, the reverse Top-k geo-social keyword query [14] and time-aware spatial keyword query [15] in the road network were studied, and two novel index structures GIM tree and TG were designed. Sheng et al. [16] discussed the Top-k paradigm trajectory search problem and proposed a framework using incremental extension and point-wise similarity. Li et al. [17] discussed such queries in the wireless broadcasting environment and designed a new air index named SKQAI to facilitate the effective processing of queries. Recently, the time-interval augmented spatial keyword query on the road network has also been discussed and processed [18].

B. WHY-NOT QUESTIONS

To improve the availability of database query results, Chapman and Jagadish [19] first proposed the why-not question. The existing methods to solve why-not questions can be divided into three categories: 1) *manipulation identification* [19]; 2) *database modification* [20]; 3) *query refinement* [21]. However, *manipulation identification* can only identify the operations that result in the exclusion of missing objects from the result set, but can not make those objects into result objects. Although *database modification* will make the missing object become the result object, in real life, users usually do not have permission to access the data in the database. *Query refinement* can not only prompt users to modify the query according to their own needs, but also make the why-not objects expected by users become the result object with the least cost. Therefore, *query refinement* is widely used to solve the why-not question of various types of queries. Chen et al. [22] discussed why-not questions on spatial keyword queries, and modified the score calculation function and the input keyword information to include missing objects in the query result. Then they dealt with the same problem by modifying the query keyword and the preference parameter between spatial distance and text similarity [23]. Chen et al. [24] studied the why-not question on direction-aware spatial keyword queries, and solved this problem by adjusting the query direction and k value. In addition, the researchers also studied the why-not question on various types of queries, such as, direction-aware spatial keyword

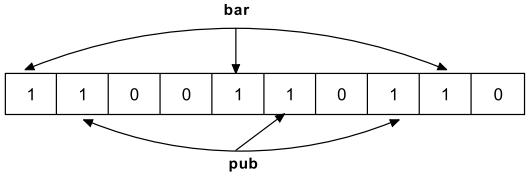


FIGURE 2. An example of bloom filter.

TABLE 2. Symbols and definitions.

Symbol	Definition
$o.doc$	textual information of object o
$o.t[o.st, o.et]$	effective time of object o, $o.st$: the start time, $o.et$: the end time
$T_tsim(q,o)$	textual similarity between query q and object o
$Dsim(q,o)$	spatial proximity between q and o
$T_isim(q,o)$	temporal similarity between q and o
$STT(q,o)$	spatial-textual-temporal similarity between q and o
$cost(q,q')$	the modification cost of the refined query q' relative to the original query q
$R(q,o)$	the ranking of object o in the query results
R_W	the lowest ranking of why objects in W

queries [25], [26], the group spatial keyword queries [27], the geo-social keyword queries in the road network [28], range-based skyline queries [29], and reverse kNN queries [30].

C. WHY QUESTION

At present, there are few research results on the why question. Gao et al. [1] discussed the why-not and why question on reverse Top-k queries. Their solution to why questions adopts the same method as the why-not questions, namely query refinement. However, there are no relevant research results to solve why questions of SKQ queries. Therefore, We first propose and study why questions of SKQ queries in traffic networks. Since the query refinement method is widely used to solve various kinds of why-not questions, we also adopt query refinement to deal with our why question.

III. PRELIMINARIES AND PROBLEM DEFINITION

This section first defines the time-aware spatial keyword query, and then formalizes the why question of such queries. Table 2 summarizes the commonly used symbols and their descriptions.

A. BLOOM FILTER

Bloom filter is an effective data structure proposed by B.H. Bloom in 1970. It can quickly determine whether an element belongs to a set $W = \{w_1,\ldots, w_n\}$. The bloom filter is composed of an m-bit binary vector and k mutually independent hash functions H_i ($i \in [0, k-1]$), which is recorded as BF(m,k). The initial state of the binary vector is 0, and the hash function is mapped to a bit of the binary vector, that is, $H_i = \{0,1\}^* \rightarrow [0, \text{m-1}]$. Fig. 2 shows an example of Bloom filter BF(m,k) with $m = 10, k = 3, W = \{\text{bar, pub}\}$ and mapping *bar* and *pub* to BF(m,k), that is, W is represented by binary vector [1,1,0,0,1,1,0,1,0] and recorded as $V^W_{BF(m,k)}$.

The process of using Bloom filter BF(m,k) to find an element t is as follows: 1) execute the above k hash functions for t to obtain k positions corresponding to the bit array; 2) If one of the k positions is 0, t must not be in the set, which is marked as $t \overset{BF(m,k)}{\rightarrow} 0$; If all k positions are 1, t may be in the set, which is marked as $t \overset{BF(m,k)}{\rightarrow} 1$.

B. TIME-AWARE SPATIAL KEYWORD QUERIES IN TRAFFIC NETWORKS (TSKQT)

Suppose that a traffic network is an undirected weighted graph $G = (V, E)$, where V and E represent the node set and edge set of the traffic network, respectively. A node $v \in V$ represents the intersection or terminal node of the road segment in the traffic network. An edge $e(v_i, v_j, w) \in E$ is the road segment between nodes v_i and v_j ($v_i, v_j \in V$ && $i \neq j$), and w is a non-negative weight representing the length of e.

definition 1: Spatial-textual Object

A spatial-textual object o in the traffic network can be denoted as $o = (o.loc, o.doc, o.TI)$, where $o.loc$ is the object location on an edge of the traffic network, $o.doc$ is a set of keywords describing this object, which is formally defined as $o.doc = \{key_1, key_2, \ldots, key_n\}$. $o.TI$ is the effective time interval of o, modeled as $o.TI = [o.st, o.et]$, where $o.st$ and $o.et$ are the start and end time of o, respectively.

definition 2: Query point in traffic network

A query q in a traffic network can be denoted as $q = (q.loc, q.doc, q.TI)$, where $q.loc$, $q.doc$, and $q.TI$ have the same meaning as $o.loc$, $o.doc$, and $q.TI$.

definition 3: Keyword Matching

Given a query q and an object o, q and o are keyword matching if and only if they share one or more keywords, i.e., $q.doc \bigcap o.doc \neq \oslash$.

definition 4: Temporal Matching

Query q and object o are temporal matching if and only if there is an intersection between their time intervals, i.e., $q.TI \bigcap o.TI \neq \oslash$.

definition 5: Comprehensive Matching

Query q and object o are comprehensive matching if and only if they are both keyword and temporal matching, i.e., $q \overset{cm}{\rightarrow} o = (q.doc \overset{km}{\rightarrow} o.doc) \wedge (q.TI \overset{tm}{\rightarrow} o.TI)$.

Hereinafter, $q.doc \overset{km}{\rightarrow} o.doc$, $q.TI \overset{tm}{\rightarrow} o.TI$, and $q \overset{cm}{\rightarrow} o$ are used to represent keyword, temporal, and comprehensive matching, respectively.

Next, our scoring function is given below to calculate the comprehensive similarity between query q and object o as in (1), shown at the bottom of the next page, where $0 \leq \alpha, \beta, \gamma \leq 1$ are query preference parameters, which represent users' preferences for spatial proximity, textual similarity, and temporal similarity, respectively, and $\alpha + \beta + \gamma = 1$. To simplify the discussion, the values of these three parameters are assumed to be $\frac{1}{3}$. The spatial proximity between o and q is denoted as $Dsim(q, o)$. We know that the Sigmoid

function value changes quickly at small variables and slowly at large variables, which conforms to users' intuition, that is, short-distance driving is usually more sensitive to distance than long-distance driving. Thus, the sigmoid function is used to standardize the spatial proximity to the range [0,1], where $\rho \in (0.1]$ is the distance adjustment parameter, and $D_N(q, o)$ is the network distance from q to o.

$$Dsim(q, o) = 2 - \frac{2}{1 + e^{-\rho \times D_N(q,o)}}, \tag{2}$$

The textual similarity $T_tsim(q, o)$ can be calculated by the Jaccard function, language model, and Cosine similarity function. In particular, the ratio of the sum of the weights of the keywords common to the query and the object to the sum of the weights of query keywords is used to calculate $T_tsim(q,o)$, as shown in Equ. 3.

$$T_tsim(q, o) = \begin{cases} \frac{\sum_{t \in (q.doc \cap o.doc)} t.w}{\sum_{t \in q.doc} t.w}, & others \\ -\infty, & if\ o.doc \cap q.doc = \phi, \end{cases} \tag{3}$$

where the weight of each keyword t is calculated according to its occurrence frequency in the system, and the global ranking of keywords is given according to the decreasing order of the weight.

The temporal similarity $T_isim(q, o)$ can be expressed by the overlap rate between the time interval of q and the time interval of o, as shown in Equ. 4 below.

$$T_isim(q, o) = \begin{cases} \frac{|q.TI \cap o.TI|}{|q.TI|}, & others, \\ -\infty, & if\ q.TI \cap o.TI = \phi \end{cases} \tag{4}$$

The comprehensive similarity score of the object can be calculated through the above scoring function (Equ. 1). The higher the score of the object, the higher its ranking. In particular, the ranking of object o can be obtained from its score as follows:

$$R(q, o) = |\{o' \in O | STT(q, o) < STT(q, o')\}| + 1, \tag{5}$$

definition 6: Time-aware Spatial Keyword Queries in the traffic network, *TSKQT* for short.

Given an object set O, a *TSKQT* query $q = (q.loc, q.doc, q.TI, q.k)$ retrieves a set RS containing k objects from O, such that $\forall o \in RS \ \&\& \ \forall o' \in O - RS$, STT($q$, o')<STT(q, o).

C. THE WHY QUESTION ON TSKQT QUERIES

1) WHY OBJECTS AND REFINED QUERIES

When the user issues a TSKQT query q =($q.doc$,$q.loc$, $q.TI$,$q.k$) in the traffic network, he may find that the result set includes some unexpected objects, which are called why objects $W = \{w_1, w_2, \ldots, w_j\}$. Then, the user may initiate a subsequent why question with the why object set W to find a refined query $q' = (loc, doc', TI', k')$, whose result set does not contain any why objects.

Assuming that all objects and queries are stationary, and users' preferences for the network distance, textual similarity, and temporal matching are fixed during query processing. In order to ensure that all unexpected objects are discarded from the result set of the low-cost refined query, while preserving as many other original result objects as possible, candidate refined queries can be obtained by changing the query keyword set, query time interval, and value k.

2) COST FUNCTION

By modifying the query keyword set $q.doc$, the query time interval $q.TI$, and the number of result objects $q.k$, many qualified refined queries that can exclude why objects may be generated. The refined query with the least modification relative to the original query is preferred. Therefore, a cost function is defined to quantify the modification cost of the refined query. Specifically, it is expressed as the weighted sum of Δk, Δdoc, and ΔTI, where Δk, Δdoc, and ΔTI are the modification to $q.k$, $q.doc$, and $q.TI$, respectively. For a refined query q', its modification cost relative to the original query q can be formally defined as follows:

$$cost(q, q') = \delta_1 \cdot \frac{\Delta k}{k_0 - R(W, q) + 1} + \delta_2 \cdot \frac{\Delta doc}{\sum_{t \in (q.doc \cup W.doc)} t.w} + \delta_3 \cdot \frac{\Delta TI}{\Delta TI_{max}} \tag{6}$$

where δ_1, δ_2, and δ_3 are user preference parameters for modifying $q.k$, $q.doc$, and $q.TI$, respectively. $\delta_1, \delta_2, \delta_3 \in [0,1]$ and $\delta_1 + \delta_2 + \delta_3 = 1$. For q', the ranking of the why objects $R(q', W) = min_{w_i \in W} R(q', w_i)$, the number of query results $k' = R(q', W)$-1, and Δk (the modification of k) $= |min(0, k' - k_0)|$, where k_0 is the k value of q. A basic refined query q_b is to reduce k to $R(q, W)$-1, while retaining other parameters of the original query; For other refined queries obtained by modifying query parameters and whose cost is lower than that of q_b, Δk does not exceed $k_0 - R(q, W) + 1$. Similar to the why-not question, Δk is normalized with $k_0 - R(q, W) + 1$.

Δdoc is the sum of the weights of the changed keywords adjusted from $q.doc$ to $q'.doc'$, $W.doc = \cup_{i=1}^{j} w_i.doc$, and we normalize Δdoc with $\sum_{t \in (q.doc \cup W.doc)} t.w$. The why object can be excluded from the result set by modifying the query start time and end time, and ΔTI represents the time modification from $q.t$ to $q'.t'$. Note that modifying the start and/or end time of the query may change the length and position of

$$STT(q, o) = \begin{cases} \alpha \cdot Dsim(q, o) + \beta \cdot T_tsim(q, o) + \gamma \cdot T_isim(q, o) & q \overset{cm}{\rightarrow} o; \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

the original query interval, so ΔTI is represented by Equ. 7 below.

$$\Delta TI = \xi \cdot \left(\left|\frac{q'.et - q'.st}{2} - \frac{q.et - q.st}{2}\right|\right) + (1-\xi) \cdot \left|(q'.et - q'.st) - (q.et - q.st)\right| \quad (7)$$

The first part of Equ. 7 represents the interval position shift, the second part represents the length modification of the time interval, and ξ is used to balance the importance of interval position shift and interval length modification. ΔTI_{max} represents the maximum modification of the time interval, which is used to normalize ΔTI to the range [0, 1]. In particular, $\Delta TI_{max} = \xi \cdot (|\frac{max(W.et) - min(W.st)}{2} - \frac{q.et - q.st}{2}|) + (1-\xi) \cdot |(max(W.et) - min(W.st)) - (q.et - q.st)|$.

definition 7: **The why question of time-aware spatial keyword queries,** *WhyTSKQT*.

Given an original time-aware spatial keyword query $q = (q.loc, q.doc, q.TI, q.k)$, and a why object set $W = \{w_1, w_2, \ldots, w_j\}$, the why question on time-aware spatial keyword queries aims to return a refined query $q' = (q'.loc, q'.doc, q'.TI, q'.k)$ with the lowest penalty cost according to Equ. 6, so that any why object $w_i \in W$ can be discarded from the result set of the refined query q'.

IV. TTG-TREE INDEX

Like other similar studies, the traffic network is represented by a weighted graph composed of a vertex set and an edge set. In addition, the traffic network also includes a group of objects and *TSKQT* queries distributed on its edges. Due to the large number of data objects in the traffic network, the network space and objects irrelevant to the *TSKQT* query should be pruned as much as possible. To this end, a new hybrid index TTG-tree is proposed, which can support distance pruning, text pruning and time pruning at the same time. As shown in Fig.3, TTG-tree is mainly composed of three parts, namely, the G-tree & its DM Metrices of the traffic network, the Keyword list & the Bloom buffer for distinguishing, and the Text&Time part.

A. THE G-TREE & ITS DM METRICES OF THE TRAFFIC NETWORK

G-tree [31] is an assembly-based index which can efficiently support location-based queries on the traffic network. In particular, we use the multi-level partitioning algorithm [32] to recursively partition the traffic network (graph G) into subgraphs of equal size and minimize the number of border vertices at the same time (the partition result of the traffic network in Fig. 1 is shown in Fig. 4), and use a G-tree to keep the partition result (as shown in Fig. 3 (a)). Then we precalculate and keep the distance metrics (DM) which include the shortest-path distance between border-border pairs (or border-vertex pairs), so as to accelerate the distance calculation during query processing. As shown in Fig. 3 (a), the DM of each subgraph (or graph) is also given.

B. KEYWORD LIST & BLOOM BUFFER FOR DISTINGUISHING

Each keyword in the system is given a weight according to the frequency of occurrence, and a global ranking is given according to the order of weight from large to small. In addition, keywords are divided into frequent keywords and infrequent keywords according to the occurrence frequency, which can be distinguished by a bloom filter, as shown in Fig. 3(b).

As discussed in Section III-A, the bloom filter is composed of an m-bit binary vector and k mutually independent hash functions H_i ($i \in [0, k-1]$), which is recorded as BF(m,k). It can quickly determine whether an element belongs to a set $W = \{w_1, \ldots, w_n\}$. However, the bloom filter may give some false positives, that is, it is possible to judge the elements that do not belong to a set as belonging to the set. For BF(m,k) and set $W = \{w_1, \ldots, w_n\}$, the false positive rate is $(1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-\frac{kn}{m}})^k$, and when $k = (n/m)ln2$, the misjudgment rate is the lowest, equal to $(1 - \frac{1}{2})^k = 2^{-ln2(n/m)} \approx 0.618^{m/n}$. For example, when $m/n = 9$, the misjudgment rate is 0.013.

C. TEXT&TIME PART FOR SUBGRAPH (NODE)

Finally, we construct the Text&Time part as shown in Fig. 3 (c), i.e., the textual & temporal index of the objects in the node.

For each non-leaf node c_i, we keep the following items:

- the *id* of node c_i;
- pointers to the child-nodes of c_i.
- the bloom filter for the frequent keywords contained by the objects in the nodes c_i, which can be used to quickly determine whether a frequent keyword is included in c_i. Considering the large number of objects containing frequent keywords in each non-leaf nodes, in order to save storage space and comparison time, a bloom filter is used to represent the keyword inclusion of objects in the node.
- inverted lists of infrequent keywords (IL_k). This is because the number of nodes and objects satisfying infrequent keywords is relatively small, and the inverted file occupies relatively small storage space. Moreover, considering that the bloom filter has a certain false-positive rate, the inverted file is directly used to improve the filtering efficiency, taking into account the space utilization and pruning effect.
- inverted lists of time intervals (IL_t). Specifically, a day is divided into 24-hour intervals. For an hour interval, if node c_i contains objects open for business during this period, the list of sub-nodes containing those open objects is recorded.

For each leaf node c_i, the following items are kept, i.e.,

- the *id* of c_i;
- inverted lists (IL_k) of all the keywords contained in c_i.
- inverted lists of time intervals (IL_t). In particular, a list of objects opened during each hour interval is recorded.

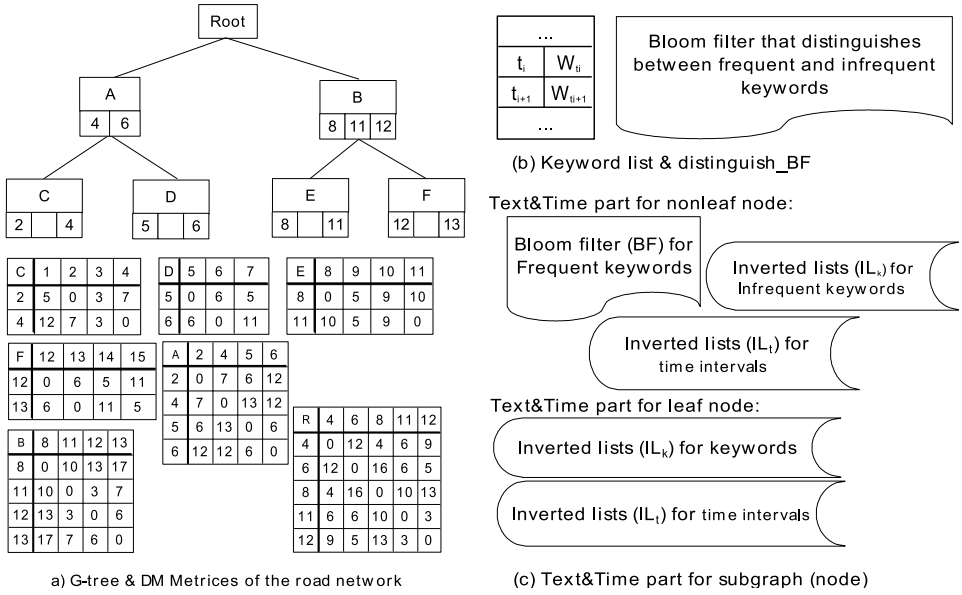


FIGURE 3. The structure of the *TTG*-tree Index.

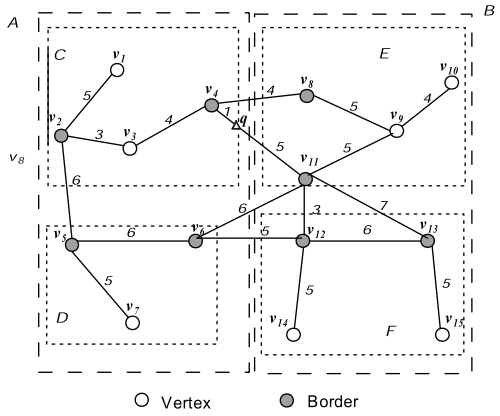


FIGURE 4. The partition result of the traffic network in Fig.1.

V. *TTG*-TREE INDEX BASED METHOD

This section discusses the *TTG-tree* based solution to *WhyTSKQT* queries in detail.

A. PRUNING TECHNIQUES

Firstly, we proposed four lemmas to prune the irrelevant search space and ineligible objects of the traffic network in an efficient way.

lemma 1: For a *TSKQT* query $q = (q.loc, q.doc, q.TI, q.k)$ and a node c_i of TTG-tree, c_i can be safely pruned if

$$d_N^{Low}(q, c_i) > -\frac{1}{\rho}\ln(\frac{2}{2 - \frac{STT(q,o_k)-\beta-\gamma}{\alpha}} - 1),$$

where o_k is the top-k result object of q, and $STT(q, o_k)$ is the comprehensive similarity score between o_k and q.

Proof: It is proved by contradiction. Suppose

$$d_N^{Low}(q, c_i) > -\frac{1}{\rho}\ln(\frac{2}{2 - \frac{STT(q,o_k)-\beta-\gamma}{\alpha}} - 1)$$
$$\&\& \exists o \in c_i, STT(q, o) > STT(q, o_k).$$

Thus for any object o in c_i, certainly

$$d_N^{Low}(q, o) > -\frac{1}{\rho}\ln(\frac{2}{2 - \frac{STT(q,o_k)-\beta-\gamma}{\alpha}} - 1).$$

Through transformation, we have

$$Dsim(q, o) < \frac{STT(q, o_k) - \beta - \gamma}{\alpha}.$$

According to Equ. 2, we know that

$$STT(q, o) = \alpha \times Dsim(q, o) + \beta \times T_esim(q, o) + \gamma \times T_isim(q, o),$$

and since

$$T_esim(q, o), T_isim(q, o) \in [0, 1],$$

we have

$$STT(q, o) \leq \alpha \times Dsim(q, o) + \beta + \gamma.$$

Thus,

$$STT(q, o) < \alpha \times \frac{STT(q, o_k) - \beta - \gamma}{\alpha} + \beta + \gamma < STT(q, o_k),$$

which contradicts with our hypothesis. As a result, c_i can be safely ignored. ■

lemma 2: For a *TSKQT* query $q = (q.loc, q.doc, q.TI, q.k)$ and a node c_i of TTG-tree, if $c_i.IL_t.TI \cap q.TI = \phi$,

where $c_i.IL_t.TI$ is the union of time intervals having a non-empty list in $c_i.IL_t$, c_i will be pruned.

Proof: Suppose for node c_i, $c_i.IL_t.TI \cap q.TI = \phi$ and c_i includes the result object o. Thus, $q.TI \overset{tm}{\rightarrow} o.TI$, which means $o.TI \cap q.TI \neq \phi$. Certainly, $c_i.IL_t.TI \cap q.TI \neq \phi$, which contradicts the hypothesis. ■

lemma 3: For a *TSKQT* query $q = (q.loc, q.doc, q.TI, q.k)$ and a node c_i of TTG-tree, if $\forall(frequent\ key_i \in q.doc)\ key_i \overset{BF(m,k)}{\rightarrow} 0$ && $\forall(infrequent\ key_i \in q.doc)\ !\exists c_i.IL_k.key_i)$, then node c_i will be pruned.

Proof: Assume that $\forall(frequent\ key_i \in q.doc)\ key_i \overset{BF(m,k)}{\rightarrow} 0$ && $\forall(infrequent\ key_i \in q.doc)\ !\exists c_i.IL_k.key_i$, and c_i includes the result object o. Thus, $q.doc \overset{km}{\rightarrow} o.doc$, certainly, $\exists key_i \in o.doc$. If key_i is a frequent keyword, we have $key_i \overset{BF(m,k)}{\rightarrow} 1$; otherwise $c_i.key_i.IL_k$ exists. This contradicts the hypothesis, thus the lemma is proved. ■

1) UPPER BOUND SIMILARITY CALCULATION

In view of the relatively limited pruning intensity of these above lemmas, to further reduce the irrelevant region, $STT^{UB}(q, c_i)$ is calculated for each node c_i:

- To calculate $Dsim^{UB}(q, c_i)$, if q is not within c_i, the shortest network distance from q to the boundary of c_i is used; otherwise $Dsim^{UB}(q, c_i)$ equals 0.
- To calculate $T_tsim^{UB}(q, c_i)$, we divide the sum of weight values of all query keywords existing in c_i by the sum of weight values of query keywords, that is, $T_tsim^{UB}(q, c_i) = \frac{\sum_{t \in (q.doc \cap c_i.IL_k)} t.w}{\sum_{t \in q.doc} t.w}$. Note that for non-leaf node c_i, if key_i is a frequent keyword, the bloom filter in c_i is used for judgment; Otherwise, the inverted lists will work. If c_i is a leaf node, it is all judged by the inverted file.
- To calculate $T_isim^{UB}(q, c_i)$, we use the ratio of the overlapping length between the effective time interval of c_i and the query interval to the query interval length, i.e., $T_isim^{UB}(q, c_i) = \frac{|c_i.IL_t.TI \cap q.TI|}{|q.TI|}$.

Finally, $STT^{UB}(q, c_i) = \alpha \times Dsim^{UB}(q, c_i) + \beta \times T_tsim^{UB}(q, c_i) + \gamma \times T_isim^{UB}(q, c_i)$.

lemma 4: For a *TSKQT* query $q = (q.loc, q.doc, q.TI, q.k)$ and a node c_i of TTG-tree, if $STT^{UB}(q, c_i) < STT(q, o_k)$, where o_k the current top-k result object of q, and $STT^{UB}(q, c_i)$ is the upper bound of the comprehensive similarity between q and any object in c_i, c_i is safely pruned.

Proof: For any object o in c_i, because $STT^{UB}(q, c_i) < STT(q, o_k)$, at least k objects are more similar to query q than o. Thus, o cannot be one of the top-k result objects.

■

B. THE BASIC WhyTSKQT QUERY PROCESSING METHOD

The goal of this subsection is to obtain candidate refined queries by modifying query keywords and the query time interval, thus obtaining the least expensive refined query. Its result set does not contain any why objects, and retains as many original query result objects as possible. For each refined query to be explored, the above lemmas will be used to reduce massive ineligible objects.

1) PREPARATION OF CANDIDATE QUERY KEYWORD SETS AND CANDIDATE QUERY TIME INTERVALS

Considering that adding a keyword unrelated to the original query result set may exclude non-why result objects and insert more new result objects, we only consider the keywords belonging to the original result objects. Firstly, the keywords of all the original result objects are divided into three subsets, namely $SK_{\overline{W}R}$, SK_{WR} and $SK_{W\overline{R}}$. $SK_{W\overline{R}}$ represents the set of keywords contained only by the why objects but not by other result objects, SK_{WR} includes the keywords contained by both the why objects and other result objects, and $SK_{\overline{W}R}$ is the set of keywords not contained by the why objects but by other result objects, i.e., $SK_{W\overline{R}} = \biguplus_{o \in W} o.doc - \biguplus_{o' \in q.results - W} o'.doc$

$SK_{WR} = \biguplus_{o \in q.results - W} o.doc \bigcap \biguplus_{o' \in W} o'.doc$;

$SK_{\overline{W}R} = \biguplus_{o \in q.results - W} o.doc - \biguplus_{o' \in W} o'.doc$

Then, we use the list AKL to keep the keywords to be added, which is an ordered list of keywords in $SK_{\overline{W}R}$, arranged in descending order by keyword weight. Similarly, the list DKL is used to keep the keywords to be deleted. It consists of two parts, the first part is an order list of keywords in $SK_{W\overline{R}}$, and the second one is an order list of keywords in SK_{WR}. Both $SK_{W\overline{R}}$ and SK_{WR} are arranged in descending order by keyword weight. These two lists are built before the query processing, and the checking order of candidate keywords is very important to obtain the least expensive refined query in an efficient manner.

As for the time interval of the refined query ($q'.TI$), we will discuss how to obtain the appropriate $q'.TI$ by modifying the original query time interval $q.TI$ in the following four cases. The ordered list QTL is used to keep the candidate query intervals.

- Case 1: when $W.st < q.st < W.et < q.et$. According to the previous discussion, the smaller the overlap rate between the time intervals of query q and the why objects of W, the lower the similarity between q and W, and the more likely why objects are excluded from the query result set. Therefore, the overlap rate between the time intervals of q and W can be reduced by increasing the query interval length or reducing the length of the interval where the query interval intersects with the time interval of W (i.e. increasing the query start time). Considering that the query modification of the user's intention should be minimized as much as possible, the method of fixing the query time length (i.e., $q.et$-$q.st$) and synchronously increasing $q.st$ and $q.et$ is adopted here. When $q.st$ increases to approach $W.et$, the overlap rate between the time intervals of the query and why objects decreases. Therefore, in case 1, the value range of the query start time is $(q.st, W.et]$, and the query end time changes synchronously. In order to facilitate the enumeration of $q.st$, 0.5 h is used as the step to increase

the value of $q.st$. In this way, we can get an ordered query time list QTL, i.e., $\{[q.st, q.et], [q.st+0.5, q.et+0.5], \ldots, [q.st+(W.et-q.st), q.et+(W.et-q.st)]\}$.

- Case 2: when $W.st < q.st$ and $W.et > q.et$, i.e. $q.TI \in W.TI$. If the method of reducing $|q.TI|$ is adopted, not only the ranking of W will not be increased, but also $\Delta t > 0$, which increases the cost of query modification; If $|q.TI|$ is increased, 1) the number of objects retrieved for query processing will be increased; 2) since $q.TI$ is included in $W.TI$, the increase of $|q.TI|$ will not have a higher impact on the similarity of why objects than on non-why result objects, so it will not increase the ranking of W. In conclusion, reducing or increasing $|q.TI|$ will not get the optimal refined query q', so the query time remains unchanged and is fixed to the original query time. At this time, QTL contains only one element, that is, the original query time $q.TI$.
- Case 3: when $q.st < W.st < q.et < W.et$. It is similar to Case 1, but the processing method is just the opposite of Case 1. That is, we fix the query time length and reduce the value of query end time. The value range of query end time is $(q.et, W.st]$, and the query start time changes synchronously. In this way, we can got the query time list QTL, i.e., $\{[q.st, q.et], [q.st-0.5, q.et-0.5], \ldots, [q.st-(q.et-W.st), q.et-(q.et-W.st)]\}$.
- Case 4: when $q.st < W.st$ and $q.et > W.et$, i.e. $W.TI \in q.TI$. Similar to Case 2, reducing or increasing $|q.TI|$ will not obtain the optimal refined query. Therefore, the processing method is the same as that of Case 2, and the query time list QTL can be obtained.

2) TTG-TREE BASED ALGORITHM

Algorithm 1 details the TTG-tree based method to deal with the why question on TSKQT queries. It outputs the least expensive refined query q', and takes as inputs the TTG-tree index, the original query q, the why object set W, the penalty p_b of the basic refined query q_b, the cardinality k_m of the result set of q_b, and the lowest ranking R_W of why objects among all the result objects of q_b. As for p_b, it equals cost(q,q_b) by using equation 6. Recall that q_b is the basic refined query discussed before.

Queue $\mathcal{D}$, pointer $TNode$, and set RRS, are initialized to empty, to keep the qualified TTG-tree nodes (subgraphs), the tree node being accessed, and the objects satisfying the refined query requirement, respectively. The variable p_c, whose initial value is p_b, is used to keep the cost value of the current best-refined query.

Remember that we have built three lists AKL, DKL, and QTL beforehand. Next, the keyword set $q'.doc$ of the refined query q' is initialized to $q.doc_0$. In the following, each candidate refined query q' is obtained by parameter modification, and these refined queries are explored to obtain the least expensive refined query until all three lists are empty. In particular, the first keyword in AKL is extracted and added to $q'.doc$, and the first keyword in DKL is taken out and deleted from $q'.doc$. The first element in QTL is taken as the query time of q', i.e., $q'.TI$, and k' is set to be k_0. Here function $DeQueue()$ gets and returns its first element. Also, the cost of q', $cost(q, q')$, is computed according to Equ. 6, so as to eliminate the refined queries with a higher cost than p_c as soon as possible. If $cost(q, q')$ is less than p_c, we start the process of q' by initializing $STT(q', o_k)$ to zero and locating the leaf node (subgraph) where q is located.

For each object o in leaf(q), it pushes o together with its $STT(q', o)$ into heap $\mathcal{D}$ if its STT value is not less than $STT(q', o_k)$, and updates $STT(q', o_k)$ accordingly. Then, it uses pointer $Tnode$ to keep the upper-most node (subgraph) visited of $\mathcal{D}$ and uses variable P^{UB} to keep the upper bound of its STT score, i.e., $STT^{UB}(q', Tnode)$. We let $Tnode$ point to leaf(q) and P^{UB} be $STT^{UB}(q', Tnode)$, and then visit the $TTG\text{-}tree$ in a bottom-up manner. If $\mathcal{D}$ is empty, the Regulate function (shown in Algorithm 2) is called to move $Tnode$ to its parent node and update P^{UB} accordingly.

Next, a tuple (c, stt) is popped-out from heap $\mathcal{D}$. Note that (c, dis) is the head element of $\mathcal{D}$, and $\mathcal{D}$ is ordered by the (upper bound of) STT scores of its elements with respect to query q. If dis, which is the (upper bound of) STT score of head element c, is smaller than P^{UB}, then the query answer may be existed in the parent node of $Tnode$, thus Regulate function is called to move $Tnode$ to its parent node and update P^{UB} accordingly. Otherwise ($dis >= P^{UB}$), there are three cases: 1) c is an object. c is a result object since it is the object with the maximum STT value; 2) c is a non-leaf subgraph. For each unvisited subgraph s of c, we use Lemmas 1, 2, 3, and 4 to check if s is a qualified subgraph. If true, s together with its upper bound STT score is inserted into $\mathcal{D}$ orderly; 3) c is a leaf node. We calculate the STT score of each qualified object o of c, and insert o together with its STT score into $\mathcal{D}$ orderly.

Let k' be the smaller of k' and $R(w, q')$. If k' is larger than R_W, the cost of q' is calculated. Then, if $cost(q, q') < p_c$, p_c is updated with $cost(q, q')$. After the qualified refined queries are processed, the lowest-cost refined query is returned.

Algorithm 2 details the $Regulate$ function. It first moves $TNode$ to its parent node. Then, for each unvisited child-node s of $TNode$, if it is not pruned by lemmas 1, 2, 3, and 4, then it is a promising subgraph and s together with its upper bound STT score is inserted into $\mathcal{D}$ orderly. Note, the value of P^{UB} is updated accordingly.

C. THE IMPROVED TTG-TREE BASED ALGORITHM

The above TTG-tree based algorithm retrieves the possible candidate keyword set and query time interval set according to the increasing order of cost influence to form the candidate refined queries, and then executes each candidate query to determine whether all the why objects are excluded from the returned query result set. Although the above algorithm only enumerates the keyword sets and query time intervals that have a positive effect on the query results, filters the refined queries obtained according to the cost value to form the candidate refined queries, and adopts several pruning techniques in each candidate refined query processing to

Algorithm 1 TTG-Tree Based Algorithm

Input: the TTG-tree, q, W, p_b, k_m, R_W;
Output: the lowest-cost refined query $q' = (loc, doc', TI', k')$;

begin
  Init_Queue(D, D');$TNode$, RRS=∅; $p_c = p_b$;
  obtain lists AKL, DKL, and QTL; $q'.doc = q.doc_0$;
  while $AKL \neq \emptyset || DKL \neq \emptyset || QTL \neq \emptyset$ **do**
    $q'.doc = q'.doc \bigcup$ DeQueue(AKL) - DeQueue(DKL);
    $q'.TI$ = DeQueue(QTL); $k' = k_0$; calculate $cost(q, q')$ according to Equ.6;
    if $cost(q, q') < p_c$ **then**
      float $STT(q', o_k) = 0$; Locate the leaf node (subgraph) leaf(q) where q lies;
      for *each object* $o \in leaf(q)$ **do**
        if $STT(q', o) \geq STT(q', o_k)$ **then**
          InQueue($\mathcal{D}$, o, $STT(q', o)$);
      $TNode$=leaf(q); $P^{UB} = STT^{UB}(TNode, q')$;
      while $|RRS| < k'$ && $(D \neq \emptyset$ || $TNode \neq R_0)$ **do**
        if D=∅ **then**
          Regulate($TNode$, P^{UB}, $\mathcal{D}$);
        c, stt=D.pop();
        if $stt < P^{UB}$&$TNode \neq R_0$ **then**
          Regulate($TNode$, P^{UB}, $\mathcal{D}$);
        else
          if *c is an object* **then**
            insert c into RRS;
          else
            if *c is a non-leaf subgraph* **then**
              for *each unvisited & qualified child node* $s \in c$ **do**
                InQueue($\mathcal{D}$, s, $STT^{UB}(q', n_s)$);
            else
              for *each qualified object* $o \in c$ **do**
                if $STT(q', o) \geq STT(q', o_k)$ **then**
                  InQueue($\mathcal{D}$, o, $STT(q', o)$);
      k'=min(k',R(q', W));
      if $k' \geq R_W$ **then**
        calculate $cost(q, q')$ according to Equ.6;
        p_c=$cost(q, q')$ if $cost(q, q') < p_c$;
  return $q' = (loc, doc', TI', k')$;

Algorithm 2 Regulate Function ($TNode$, P^{UB}, $\mathcal{D}$)

begin
  $TNode$=$TNode$.Parent;
  for *each unvisited child node s of TNode* **do**
    if *s is not pruned by lemmas 1, 2, 3, and 4* **then**
      InQueue(D, s, $STT^{UB}(q', s)$);
      P^{UB}= max($STT^{UB}(q', s)$, P^{UB});
  return $TNode$;

speed up the query processing, there is still the possibility of further optimization.

1) EFFECTIVE DIVISION OF TIME INTERVAL

Considering that people always go out to work during the day and have much fewer activities at night, most places serving people (point of interest, POI) are open during the day and less after midnight. Therefore, when building the temporal inverted list, we divide the time domain into m time periods for daytime to keep the temporal information of objects and set a longer time period for midnight. For example, set 18 time periods from 6:00 to 24:00, and a longer time period from 0:00 to 6:00.

2) EARLY TERMINATION STRATEGY FOR CANDIDATE REFINED QUERIES I

For the refined query $q' = (loc, doc', TI', k')$, according to Equ. 6, we have,

$$cost(q, q') = \delta_1 \cdot \frac{\Delta k}{k_0 - R(W, q) + 1} + \delta_2 \cdot \frac{\Delta doc}{\sum_{t \in (q.doc \cup W.doc)} t.w} + \delta_3 \cdot \frac{\Delta TI}{\Delta TI_{max}}$$

If q' is the lowest-cost refined query, then

$$cost(q, q') < cost(q, q_b) = \delta_1 \cdot \frac{\Delta k_b}{k_0 - R(W, q) + 1}$$

thus,

$$\Delta k' < \Delta k_b$$

i.e.,

$$R(q', W) > R(q, W)$$

Therefore, during the execution of q', if the ranking of the why object does not conform to Equ. 8, the processing of q' can be ended immediately, and the refined query q' can be safely excluded.

3) EARLY TERMINATION STRATEGY FOR CANDIDATE REFINED QUERIES II

Next, we further consider the impact of the modification of query keywords and query time, that is, Δdoc and ΔTI, on the lowest ranking of why objects. The basic idea is shown in Fig.5. Suppose in the original query $q = (loc, doc, TI, 6)$, $q.k_0 = 6$ and the why object ranking $R(q, W) = 4$, we project the cost function into a plane and treat each refined query as a point on the plane. The plane P_{min} passes through the point $q'_0 = (loc, doc, TI, 3)$ (the current best refined query whose cost is cost(q, q'_0). Note that, initially, it is a basic

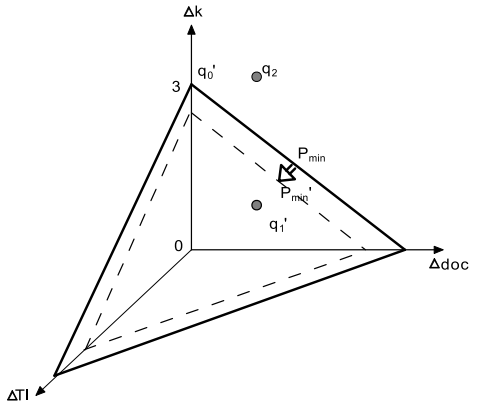


FIGURE 5. The illustrate of Early termination strategy II.

refined query q_b.), and all refining queries located on this plane have the same modification cost as q'_0. For refined queries above the plane P_{min}, such as q'_2, the modification cost is greater than P_{min}. Based on the above observations, we first determine the upper bound of Δk, that is, Δk^U, according to the current Δdoc and ΔTI, and then calculate the corresponding lowest ranking of the why objects ($R(q', W)_L$), so as to terminate the query process earlier. The details are as follows:

$$\Delta k^U = (P_{min} - \delta_2 \cdot \frac{\Delta doc}{\sum_{t \in (q.doc \cup W.doc)} t.w} - \delta_3 \cdot \frac{\Delta TI}{\Delta TI_{max}}) \cdot \frac{k_0 - R(W, q) + 1}{\delta_1}$$

$$R(q', W)_L = k_0 - \Delta k^U$$

As the query progresses, the modification cost P_{min} of the current best refined query is continuously reduced. For example, after executing q'_1, the P_{min} is reduced to $cost(q, q'_1)$ which is less than P_{min}, and the plane P_{min} is also updated to the plane P'_{min} passing through the point q'_1. As P_{min} continues to shrink, Δk^U will also become more smaller, and the lower bound of the ranking of the why object will increase with the decrease of Δk^U, which enables more refined queries to terminate earlier, thus shortening the time required to find the best refined query.

4) EARLY TERMINATION STRATEGY FOR CANDIDATE REFINED QUERIES III

During the processing of q', if the number of result objects currently retrieved reaches k_0 and the why object does not appear, the processing of the refined query q' can also be ended immediately. If $k' = k_0$, then $\Delta k = 0$. At this time, the cost(q, q') of the refined query q' is the smallest, and its value equals the sum of the last two terms of the cost function, as shown below.

$$cost(q, q') = \delta_2 \cdot \frac{\Delta doc'}{\sum_{t \in (q.doc \cup W.doc)} t.w} + \delta_3 \cdot \frac{\Delta TI'}{\Delta TI_{max}} \quad (8)$$

5) ALGORITHM OPTIMIZATION

According to the above optimization strategies, we optimize the algorithm based on TTG-tree. Specifically, lines 22-23 of Algorithm 1 are modified as follows:

if *c is an object* **then**
  if *c is a why object* && $(R(q', c) > R(q, c) || R(q', c) > R(q', W)_L)$ **then**
    break; //early termination strategy I && II
  if $R(q', c) >= k_0$ **then**
    break; //early termination strategy III
  insert c into RRS;

D. TIME COMPLEXITY ANALYSIS

Theorem 1: Suppose $|C|$ is the number of leaf nodes in the system, $|C_{remain}|$ is the number of nodes left after pruning with Lemmas 1, 2, 3, and 4, $|q.doc|$ is the average number of query keywords, $|q.TI|$ is the length of the query time interval, N_{IL_K} ($N_{IL_{TI}}$) is the average number of keyword inverted lists (hour-interval inverted lists) in a node, L_{IL_k} ($L_{IL_{TI}}$) is the average length of the inverted list for a keyword (an hour interval) of a node, and m is the size of RRS, then the time complexity of our algorithm is $O(min(max(l_1, l_2), l_3) \cdot (|C| \times (|q.doc| + |q.TI|) + |C_{remain}| \times (\frac{N_{IL_K}+1}{2} \cdot |q.doc| + \frac{N_{IL_{TI}}+1}{2} \cdot |q.TI| + \frac{|q.doc| \times L_{IL_k} + |q.TI| \times L_{IL_{TI}}}{\sqrt{w}} + (|q.doc| + |q.TI|) \cdot min(L_{IL_k}, L_{IL_{TI}})) + \tau \times log\tau + log_f \frac{|V|}{\tau} \times log_2^2 f \times |V| + mlogm))$, where l_1, l_2, and l_3 are the number of elements in AKL, DKL, and QTL, respectively.

Proof: Firstly, the loop from line 7 to 36 runs $min(max(l_1, l_2), l_3)$ times to obtain the best-refined query.

Secondly, for a refined query, it takes $O(|C| \times (|q.doc| + |q.TI|) + |C_{remain}| \times (\frac{N_{IL_K}+1}{2} \cdot |q.doc| + \frac{N_{IL_{TI}}+1}{2} \cdot |q.TI| + \frac{|q.doc| \times L_{IL_k} + |q.TI| \times L_{IL_{TI}}}{\sqrt{w}} + (|q.doc| + |q.TI|) \cdot min(L_{IL_k}, L_{IL_{TI}})))$ for obtaining the candidate result objects. The first $O(|C| \times (|q.doc| + |q.TI|))$ is for using the lemmas to prune the unpromising nodes.

For each remaining node c_i, the time cost of retrieving the candidate objects includes two parts. The first is to retrieve the matching keyword inverted lists and the matching hour-interval inverted lists, with a time complexity of $O(\frac{N_{IL_K}+1}{2} \cdot |q.doc| + \frac{N_{IL_{TI}}+1}{2} \cdot |q.TI|)$. The second part is to calculate the intersection results of up to $(|q.doc| + |q.TI|)$ (sub) inverted lists in node c_i to get the candidate results of query q, with the time complexity of $O(\frac{|q.doc| \times L_{IL_k} + |q.TI| \times L_{IL_{TI}}}{\sqrt{w}} + (|q.doc| + |q.TI|) \cdot min(L_{IL_k}, L_{IL_{TI}}))$ [33], where $|q.doc| \times L_{IL_k} + |q.TI| \times L_{IL_{TI}}$ is the upper limit of the number of distinct objects in these lists, $min(L_{IL_k}, L_{IL_{TI}})$ is the intersection size, and w is the number of binary bits of the machine word.

Thirdly, it takes $O(\tau \times log\tau + log_f \frac{|V|}{\tau} \times log_2^2 f \times |V|)$ [31] for distance computation since the G-tree is adopted to calculate the shortest path distances, and $O(mlogm)$ time is used to organize all candidate objects in descending order of STT values. Here $|V|$ is the cardinality of the vertex set in graph G, f is the fan-out of the G-tree for graph G, and τ is the maximum number of vertices per leaf of G-tree.

Overall, the time complexity is $O(min(max(l_1, l_2), l_3) \cdot (|C| \times (|q.doc| + |q.TI|) + |C_{remains}| \times (\frac{N_{IL_K}+1}{2} \cdot |q.doc| + \frac{N_{IL_{TI}}+1}{2} \cdot |q.TI| + \frac{|q.doc| \times L_{IL_k} + |q.TI| \times L_{IL_{TI}}}{\sqrt{w}} + (|q.doc| + |q.TI|) \cdot min(L_{IL_k}, L_{IL_{TI}})) + \tau \times log\tau + log_f \frac{|V|}{\tau} \times log_2^2 f \times |V| + mlogm))$. ■

VI. DISCUSSION

A. THE WHY QUESTION ON TIME-AWARE REGIONAL SKQ QUERIES IN THE TRAFFIC NETWORK

The TTG-tree based method can be used to deal with time-aware regional spatial keyword queries. Firstly, we assume that the search region of a regional query q covers all points whose network distance to the location $q.loc$ of q does not exceed a prescribed distance value $q.r$. In particular, for any node c_i of the TTG-tree, if the query region does not intersect with the traffic network area covered by c_i, c_i together with its descendant nodes can be pruned safely. Therefore, based on the above discussion, only the relevant parts of Lemma 1 and Algorithm 1 need to be modified accordingly for range query processing.

B. THE WHY QUESTION ON ORDINARY SKQ QUERIES IN THE TRAFFIC NETWORK

Our proposed method is also suitable for the why question on ordinary spatial keyword queries in traffic networks. In particular, when calculating the object similarity value and the modification cost of refined queries, the temporal similarity and time modification cost are ignored; In addition, when accessing a related node (subgraph) of the TTG-tree, the temporal inverted lists of the node are not used. Therefore, in order to process ordinary SKQ queries, we make the following two modifications: 1) delete the time-related parts of Equ. 1 and Equ. 6; 2) In the query processing of Algorithm 1, Lemma 2 and time-dependent pruning are ignored.

VII. EXPERIMENTAL STUDY

A. EXPERIMENT SETUP

1) DATASETS

In the experiments, two datasets, California (CA for short) and San Francisco Bay (SF for short),[1] are used to evaluate algorithm performance. CA consists of the traffic network of California, and a set of objects whose description information comes from GNIS.[2] For SF, we use the real traffic network data of San Francisco Bay in the United States, and object information is extracted from Twitter.[3] For each object in the traffic network, 3-5 keywords are allocated and a random location on the edge of the traffic network is given. Moreover, a gaussian-compliant time interval in the range of [0,24] is generated for each object. The detail of the datasets is showed in Table 3.

[1] www.cs.fsu.edu/ lifeifei/spatialdataset.htm
[2] geonames.usgs.gov/domestic
[3] www.twitter.com

TABLE 3. The characteristics of the dataset.

Dataset	CA	SF
$Number\ of\ vertexes$	21048	175343
$Number\ of\ edges$	21693	223308
$Number\ of\ objects$	4821	46545
$Number\ of\ keywords\ per\ object$	4.5	4.4

TABLE 4. Parameter settings.

Parameter	value (the default values are shown in bold)
k_0	6,**10**,20,30,50
$\|q.doc\|$	1,**2**,3,4
$\|q.t\|$	60,**120**,180,240 (minute)
$\|W\|$	**1**,2,3,4
$R(q, W)$	3,**5**,7,9
α, β, γ	(0.2,0.6,0.2),(0.3,0.4,0.3),**(0.4,0.4,0.2)**, (0.5,0.2,0.3)
$\delta_1, \delta_2, \delta_3, \delta_4$	0.1,0.2,**0.3**,0.4,0.5

A group of TSKQT queries are also generated, including the query location, keywords, and the query interval. Query keywords are also obtained from Twitter. The number of query keywords varies from 2 to 5, and 3 is its default value. The query time interval is within the range of [0,24] hours, and follows the Gaussian distribution. We give the performance evaluation of our methods to deal with TSKQT queries by varying the number of query keywords ($|q.doc|$), the number of query result objects (k_0), the number of Why objects ($|Why|$), the ranking $R(q, W)$ of why objects, preference parameters of similarity ($\alpha, \beta, and\ \gamma$), and preference parameters of cost function ($\delta_1, \delta_2, \delta_3, and\ \delta_4$), which are listed in Table 4.

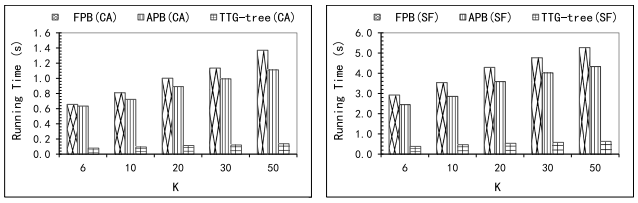
2) ALGORITHMS

As far as we know, there is no other research on the why question of time-aware SKQ queries in traffic networks in the literature. As a result, the TTG-tree method is compared with the fixed partition-based method (FPB) and the adaptive partition-based method (APB). In FPB, the traffic network is divided into grid cells of equal size; while in APB, the traffic network is divided by the adaptive partition method, so that the number of objects contained in each grid cell obtained is almost the same, and a kd-tree is used to maintain the division results. For each grid cell in FPB and APB, the textual inverted file and temporal inverted file are constructed to index the textual and temporal description of the objects within the cell. In addition, for the sake of fairness, we calculate and maintain the lower and upper distance bounds of each cell pair in the traffic network for FPB and APB methods, which facilitates distance based pruning.

B. EXPERIMENTAL RESULTS

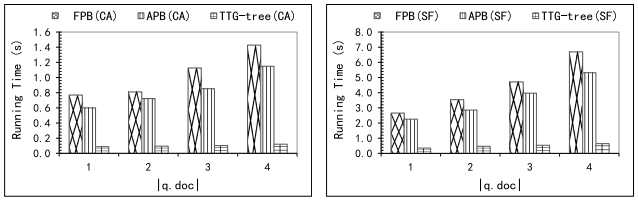
1) VARYING k_0

The first set of experiments mainly evaluate the impact of varying k_0 in the initial query on the three methods. Fig. 6 shows that the runtime of these methods increases as the value of k_0 increases, and the TTG-tree method performs



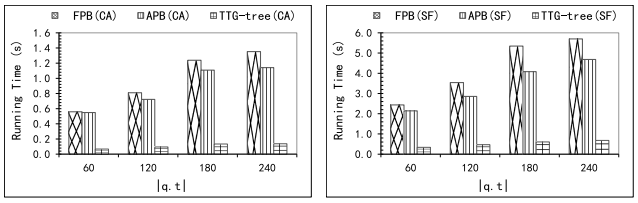
(a) CA (b) SF

FIGURE 6. Varying the value of k.



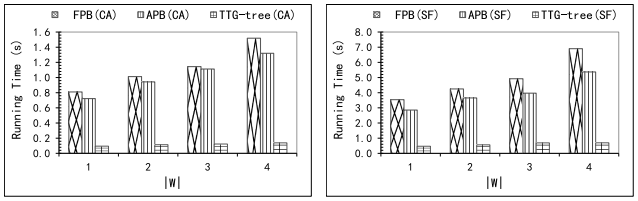
(a) CA (b) SF

FIGURE 7. Varying $|q.doc|$.



(a) CA (b) SF

FIGURE 8. Varying $|q.t|$.



(a) CA (b) SF

FIGURE 9. Varying the number of why objects ($|W|$).

better than the comparison methods. The FPB algorithm and APB algorithm perform a TSKQT query for each candidate keyword set and each candidate time interval. When the value of k_0 increases, the number of query results and the comparison and calculation required to obtain result objects increase, which increases the time cost of each TSKQT query; Secondly, the increase of k_0 value enlarges the number of candidate keyword sets and candidate time intervals, which increases the execution times of TSKQT queries. Therefore, the execution time of FPB and APB increases significantly with the change of k_0. However, due to the strategy of examining candidate refined queries in ascending order of modification cost and the early termination strategies of query processing, thus our TTG-tree is less sensitive to the change of k_0 value. In general, the running time of TTG-tree is about 12.62% of APB for the CA dataset.

2) VARYING THE NUMBER OF QUERY KEYWORDS ($|q.doc|$)

Fig. 7 shows that with the increase of the number of query keywords, the processing time required by these algorithms will also show an increasing trend. The increase of $|q.doc|$ will affect algorithm performance in two ways. First, it will increase the number of candidate keywords, thereby increasing the number of candidate keyword sets; Second, the cost spent in calculating the textual similarity will also increase. In addition, the TTG-tree method based on incremental enumeration is still superior to its competitors, and the query performance of TTG-tree is more stable than that of the comparison methods from the perspective of the growth trend of runtime.

3) VARYING THE QUERY LENGTH OF THE ORIGINAL QUERY ($|q.t|$)

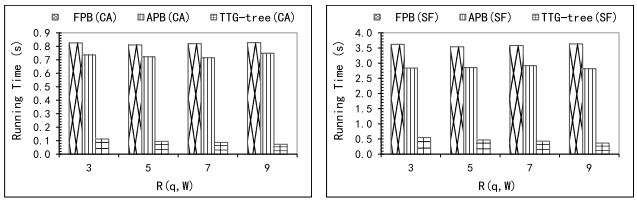
We then evaluate the impact of different query time lengths $|q.t|$ on the query performance of the methods. By observing the two groups of histograms in Fig. 8, it is found that when the query time interval becomes longer, the required time to exclude why objects in W using these algorithms also tends to increase. The reason is that the larger the $|q.t|$, the more objects that meet the TSKQT query requirements, and the higher the query cost; Second, the larger $|q.t|$, the larger the optional range of candidate time intervals and the more candidate time intervals, which increases the number of refined queries to be investigated. Fig. 8 also shows that when $|q.t|$ increases, the processing time of the TTG-tree method for excluding why objects increases slower than that of the other two methods on CA and SF data sets, indicating that TTG-tree is more stable than other two methods when modifying the length of $|q.t|$.

4) VARYING THE NUMBER OF WHY OBJECTS ($|W|$)

Then, the number of why objects is changed from 1 to 4 to verify the impact on the algorithms using CA and SF datasets. Fig. 9 shows that the total running cost of all algorithms increases gradually when $|W|$ ascends. This is because the larger the value of $|W|$ is, the larger the number of optional keyword sets and query time intervals of refined queries are, resulting in the larger number of candidate refined queries to be evaluated. When $|W|$ is small, the TTG-tree method has certain performance advantages over the other two methods; When there are more why objects, the optimization strategies of our TTG-tree scheme have a greater impact on its performance, so that the query time required by TTG-tree is much less than that of the reference methods.

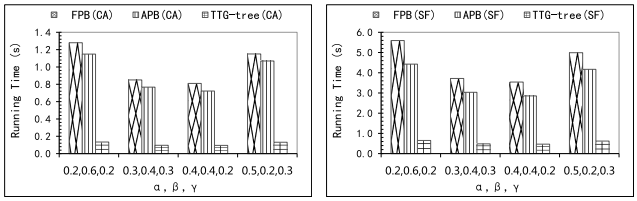
5) THE EFFECT OF $R(q, W)$

The influence of the ranking of why objects under the original query, $R(q, W)$, on these methods is also studied. Since the top-10 TSKQT query is the default setting of the original query, four queries are launched, and the ranking of the



(a) CA (b) SF

FIGURE 10. The effect of $R(q, M)$.



(a) CA (b) SF

FIGURE 11. The effect of α, β, and γ.

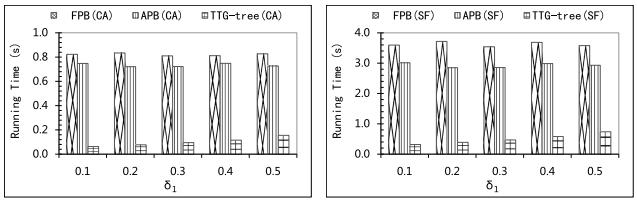
why object is 3, 5, 7 and 9, respectively. The experimental result is shown in Fig. 10. Since there is no obvious change in the candidate keyword sets and the candidate time intervals, the query time of FPB and APB methods remains almost unchanged. However, the running cost of the TTG-tree method is affected by the value of $R(q, W)$ and decreases with the increase of $R(q, W)$. The reason is that when the ranking of the why object is closer to k_0, the cost P_{min} of the initial optimal refined query will be smaller, resulting in stronger algorithm pruning ability.

6) THE EFFECT OF α, β, and γ.

We then study the influence of STT scoring function parameters on these three methods, whose results are reported in Fig. 11(a) and Fig. 11(b). Remember that α, β, and γ are the weight of spatial proximity, textual similarity, and temporal similarity on the STT scoring function, respectively. The STT function is to measure the comprehensive similarity between queries and objects. According to Equ. 6, a smaller α shows that spatial proximity does not play a great role in STT scoring function. Therefore, objects far away from the query point may also become result objects because of their high textual similarity with query keywords, the spatial pruning ability of index structures will be reduced, and a larger query space needs to be explored to find the most relevant k objects. If the value of β is small, the textual pruning ability of the algorithms will be reduced. Therefore, when α and β take the intermediate value, the query time of the three methods is less. In addition, the effective time interval of objects follows the Gaussian distribution, and the change of parameter γ has no obvious effect on algorithm performance.

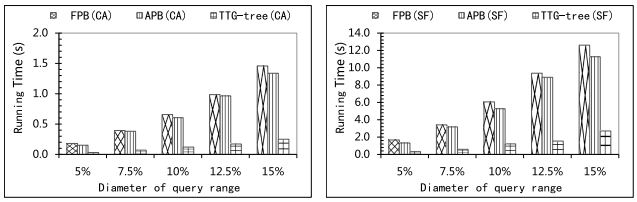
7) THE EFFECT OF δ_1, δ_2 AND δ_3

We explore the impact of cost function parameters on algorithm performance, whose result is shown in Fig. 12.



(a) CA (b) SF

FIGURE 12. The effect of δ_1.



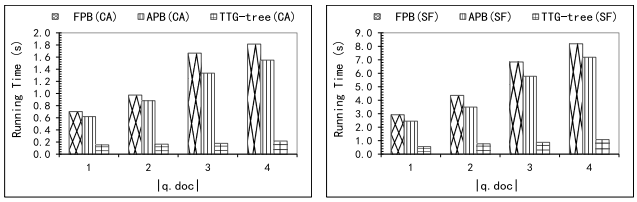
(a) CA (b) SF

FIGURE 13. Why questions of regional SKQ queries.

Note that δ_1, δ_2, and δ_3 are users' preference parameters for modifying $q.k$, $q.doc$, and $q.TI$, respectively. Since these parameters are only used to calculate the cost of the candidate refined query after executing the query to determine the ranking of why objects, δ_1, δ_2, and δ_3 have little impact on the performance of FPB and APB algorithms. However, in the TTG-tree algorithm, the basic refined query is used to initialize the optimal refined query, and the current optimal refined query together with its cost are always recorded, so as to end the query processing as soon as possible. Therefore, the performance of TTG-tree is affected by δ_1 value. According to Equ. 6, the cost of the basic refined query is δ_1, and a small δ_1 will result in a small initial P_{min} value, which can cut down the cost of algorithm execution. Therefore, the running cost of TTG-tree increases with δ_1. Since the other two parameters have little impact on algorithm performance, only the impact of δ_1 is given in Fig. 12.

8) THE PERFORMANCE ON WHY QUESTIONS OF TIME-AWARE REGIONAL SKQ QUERIES

We also evaluate the performance of these methods in dealing with the why question of time-aware regional SK queries by varying the diameter of the query region from 5.0% to 15.0% of the traffic network width. As shown in Fig. 13, the processing time of the methods shows an upward trend as the query scope expands. The reason is that the larger the query range, the more qualified objects within the query area, and the more objects need to be explored. Overall, the TTG-tree performs significantly better than its competitors. On average, for the SF dataset, the processing time of TTG-tree is only 18.87% of that of FPB.



(a) CA (b) SF

FIGURE 14. Why questions of ordinary SKQ queries.

9) THE PERFORMANCE ON WHY QUESTIONS OF ORDINARY SPATIAL KEYWORD QUERIES

Finally, we examine the performance of three methods in supporting the why questions on ordinary spatial keyword queries, without considering the temporal information of queries and objects. Fig. 14 shows that as the number of original query keywords increases, the cost of these methods also increases. Moreover, the experiment result of FPB, APB, and TTG-tree shows the same trend as that in Fig. 7, but the processing time of each method increases. The reason is that in this set of experiments, temporal filtering is not used, thus there are more qualified objects to be evaluated. Although this saves the time required for temporal matching calculation, it cannot offset the increased cost of evaluating more objects, so the total cost is still increasing.

VIII. CONCLUSION

This paper mainly focuses on the why question on time-aware spatial keyword queries in traffic networks (WhyTSKQT). In order to effectively organize the comprehensive information of the traffic network and its objects, a new hybrid index named *TTG-tree* is designed. It mainly includes three parts, the G-tree of the traffic network, the keyword list & the Bloom buffer for distinguishing, and the textual & temporal part. In addition, several pruning techniques are presented to reduce the traffic network area and massive data objects unrelated to the WhyTSKQT query. Based on TTG-tree and pruning techniques, an efficient algorithm for WhyTSKQT query processing is designed. Moreover, several optimization techniques are proposed to further speed up query processing. The experimental result on two traffic networks demonstrates the effectiveness of the proposed algorithms. In future work, the why question of spatial keyword queries on moving queries and/or objects will be considered.

REFERENCES

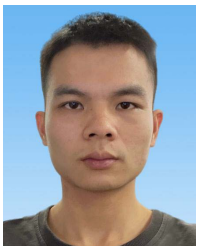
[1] Q. Liu, Y. Gao, G. Chen, B. Zheng, and L. Zhou, "Answering why-not and why questions on reverse top-k queries," *VLDB J.*, vol. 25, no. 6, pp. 867–892, Dec. 2016.
[2] I. De Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proc. IEEE 24th Int. Conf. Data Eng.*, Apr. 2008, pp. 656–665.
[3] W. Huang, G. Li, K.-L. Tan, and J. Feng, "Efficient safe-region construction for moving top-k spatial keyword queries," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, 2012, pp. 932–941.
[4] C. Zhang, Y. Zhang, W. Zhang, and X. Lin, "Inverted linear quadtree: Efficient top k spatial keyword search," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 7, pp. 1706–1721, Jul. 2016.
[5] K. Deng, X. Li, J. Lu, and X. Zhou, "Best keyword cover search," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 61–73, Jan. 2015.
[6] J. Liu, K. Deng, H. Sun, Y. Ge, X. Zhou, and C. S. Jensen, "Clue-based spatio-textual query," *Proc. VLDB Endowment*, vol. 10, no. 5, pp. 529–540, Jan. 2017.
[7] Z. Qian, J. Xu, K. Zheng, P. Zhao, and X. Zhou, "Semantic-aware top-k spatial keyword queries," *World Wide Web*, vol. 21, no. 3, pp. 573–594, May 2018.
[8] Z. Kai, S. Han, B. Zheng, S. Shang, and X. Zhou, "Interactive top-k spatial keyword queries," *Proc. VLDB Endowment*, vol. 10, no. 5, pp. 529–540, 2017.
[9] J. Yang, Y. Zhang, X. Zhou, J. Wang, H. Hu, and C. Xing, "A hierarchical framework for top-k location-aware error-tolerant keyword search," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 986–997.
[10] H. Xu, Y. Gu, Y. Sun, J. Qi, G. Yu, and R. Zhang, "Efficient processing of moving collective spatial keyword queries," *VLDB J.*, vol. 29, no. 4, pp. 841–865, Jul. 2020.
[11] J. B. Rocha-Junior and K. Nørvåg, "Top-k spatial keyword queries on road networks," in *Proc. 15th Int. Conf. Extending Database Technol.*, 2012, pp. 168–179.
[12] L. Guo, J. Shao, H. H. Aung, and K.-L. Tan, "Efficient continuous top-k spatial keyword queries on road networks," *GeoInformatica*, vol. 19, no. 1, pp. 29–60, Jan. 2015.
[13] C. Zhang, Y. Zhang, W. Zhang, X. Lin, and X. Wang, "Diversified spatial keyword search on road networks," in *Proc. EDBT*, 2014, pp. 367–378.
[14] J. Zhao, Y. Gao, G. Chen, C. S. Jensen, R. Chen, and D. Cai, "Reverse Top-k geo-social keyword queries in road networks," in *Proc. IEEE 33rd Int. Conf. Data Eng. (ICDE)*, Apr. 2017, pp. 387–398.
[15] J. Zhao, Y. Gao, G. Chen, and R. Chen, "Towards efficient framework for time-aware spatial keyword queries on road networks," *ACM Trans. Inf. Syst.*, vol. 36, no. 3, pp. 24.1–24.48, 2018.
[16] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, M. Sanderson, and X. Qin, "Answering Top-k exemplar trajectory queries," in *Proc. IEEE 33rd Int. Conf. Data Eng. (ICDE)*, Apr. 2017, pp. 597–608.
[17] Y. Li, G. Li, J. Li, and K. Yao, "SKQAI: A novel air index for spatial keyword query processing in road networks," *Inf. Sci.*, vols. 430–431, pp. 17–38, Mar. 2018.
[18] C. Luo, P. Wang, Y. Li, B. Zheng, and G. Li, "Efficient time-interval augmented spatial keyword queries on road networks," *Inf. Sci.*, vol. 593, pp. 505–526, May 2022.
[19] A. Chapman and H. V. Jagadish, "Why not?" *Sigmod*, vol. 12, no. 1, pp. 523–534, 2009.
[20] M. Herschel and M. A. Hernández, "Explaining missing answers to SPJUA queries," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 185–196, Sep. 2010.
[21] W. Zhang, Y. Li, L. Shu, C. Luo, and J. Li, "Shadow: Answering why-not questions on top-k spatial keyword queries over moving objects," in *Proc. DASFAA*, 2021, pp. 738–760.
[22] L. Chen, J. Xu, X. Lin, C. S. Jensen, and H. Hu, "Answering why-not spatial keyword top-k queries via keyword adaption," in *Proc. IEEE 32nd Int. Conf. Data Eng. (ICDE)*, May 2016, pp. 697–708.
[23] L. Chen, J. Xu, C. S. Jensen, and Y. Li, "YASK: A why-not question answering engine for spatial keyword query services," *Proc. VLDB Endowment*, vol. 9, no. 13, pp. 1501–1504, Sep. 2016.
[24] L. Chen, Y. Li, J. Xu, and C. Jensen, "Towards why-not spatial keyword top-*k* queries: A direction-aware approach," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 4, pp. 796–809, Nov. 2018.
[25] L. Chen, Y. Li, J. Xu, and C. S. Jensen, "Towards why-not spatial keyword top-*k* queries: A direction-aware approach," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 4, pp. 796–809, Apr. 2018.
[26] Y. Li, W. Zhang, C. Luo, X. Du, and J. Li, "Answering why-not questions on top-k augmented spatial keyword queries," *Knowl.-Based Syst.*, vol. 223, Jul. 2021, Art. no. 107047.
[27] B. Zheng, K. Zheng, C. S. Jensen, N. Q. V. Hung, H. Su, G. Li, and X. Zhou, "Answering why-not group spatial keyword queries," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 1, pp. 26–39, Jan. 2020.
[28] J. Zhao, Y. Gao, G. Chen, and R. Chen, "Why-not questions on top-k geo-social keyword queries in road networks," in *Proc. IEEE 34th Int. Conf. Data Eng. (ICDE)*, Apr. 2018, pp. 965–976.
[29] X. Miao, Y. Gao, S. Guo, and G. Chen, "On efficiently answering why-not range-based skyline queries in road networks (extended abstract)," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 2131–2132.

[30] Z. Zhong, X. Lin, and L. He, "Answering range-based reverse kNN and why-not reverse kNN queries," *Frontiers Comput. Sci.*, vol. 14, no. 1, pp. 233–235, Feb. 2020.
[31] R. Zhong, G. Li, K.-L. Tan, and L. Zhou, "G-tree: An efficient index for KNN search on road networks," in *Proc. 22nd ACM Int. Conf. Conf. Inf. Knowl. Manage. (CIKM)*, 2013, pp. 39–48.
[32] G. Karypis and V. Kumar, "Analysis of multilevel graph partitioning," in *Proc. ACM/IEEE Conf. Supercomputing (CDROM)*, 1995, p. 29.
[33] B. Ding and A. C. König, "Fast set intersection in memory," *Proc. VLDB Endowment*, vol. 4, no. 4, pp. 255–266, Jan. 2011.

XIAO JIA received the M.S. degree in computer application technology from the Tianjin University of Technology, in 2010, and the Ph.D. degree in biomedical engineering from Southern Medical University, in 2019. He is currently a Lecturer at the School of Computer Science and Technology, Nanyang Normal University, China. His research interests include biological information processing, spatial information and communication, and mobile computing.

YANHONG LI received the Ph.D. degree from the Huazhong University of Science and Technology (HUST), China, in 2011. She is currently an Associate Professor at the College of Computer Science, South-Central Minzu University, China. Her research interests include spatial information and communication and multimedia network technology.

YIHUA LAN received the Ph.D. degree in computer science and technology from the Huazhong University of Science and Technology (HUST), in 2011. Then, he worked as a Postdoctoral Fellow with the School of Computer Science and Technology, HUST, for four years. He is currently a Professor at the School of Computer Science and Technology, Nanyang Normal University, China. His research interests include cloud computing, machine learning, and computer-aided diagnosis.



JINLIANG HUANG received the bachelor's degree in computer science, in 2018. He is currently pursuing the master's degree with the College of Computer Science, South-Central Minzu University, China. His research interests include spatial information and communication and multimedia network technology.

XINGANG ZHANG received the master's degree in computer application technology from the Huazhong University of Science and Technology, in 2010. He is currently an Associate Professor at the School of Computer Science and Technology, Nanyang Normal University, China. He is also a Senior Member of China Computer Federation.