Fog-Computing-Based Approximate Spatial Keyword Queries With Numeric Attributes in IoV

Yanhong Li, Rongbo Zhu, *Member, IEEE*, Shiwen Mao, *Fellow, IEEE*, and Ashiq Anjum, *Member, IEEE*

***Abstract*—Due to the popularity of onboard geographic devices, a large number of spatial–textual objects are generated in the Internet of Vehicles (IoV). This development calls for approximate spatial keyword queries with numeric attributes in IoV (A^{2}SKIV), which takes into account the locations, textual descriptions, and numeric attributes of spatial–textual objects. Considering large amounts of objects involved in the query processing, this article comes up with the idea of utilizing vehicles as fog-computing resource and proposes the network structure called FCV, and based on which the fog-based top-*k* A^{2}SKIV query is explored and formulated. In order to effectively support network distance pruning, textual semantic pruning, and numerical attribute pruning, simultaneously, a two-level spatial–textual hybrid index STAG-tree is designed. Based on STAG-tree, an efficient top-*k* A^{2}SKIV query processing algorithm is presented. The simulation results show that our STAG-based approach is about 1.87× (17.1×, resp.) faster in search time than the compared ILM (DBM, resp.) method, and our approach is scalable.**

***Index Terms*—Approximate spatial keyword query (ASKQ), fog computing, Internet of Vehicles (IoV), numeric attribute, traffic network.**

I. Introduction

As an important paradigm to realize intelligent transportation system, Internet of Vehicles (IoV) enables vehicles to communicate with road-side units (RSUs) and remote cloud servers [1]. For real-time perception and geographic distribution, cloud computing is clearly not the best choice to provide communication and computing resources, since it is completely centralized [2]. Fog computing, by contrast, complements cloud computing by extending computing and caching capabilities to the edges of the network, and it facilitates localization decisions and rapid response.

Manuscript received August 1, 2019; revised November 16, 2019; accepted December 30, 2019. Date of publication January 10, 2020; date of current version May 12, 2020. This work was supported in part by the National Science Foundation under Grant 61309002 and Grant 61772562; in part by the Hubei Provincial Natural Science Foundation of China for Distinguished Young Scholars under Grant 2017CFA043; in part by the Hubei Provincial Natural Science Foundation of China under Grant 2017CFB135; in part by the Key Project of Hubei Provincial Science and Technology Innovation Foundation of China under Grant 2018ABB1485; in part by the Fundamental Research Funds for the Central Universities under Grant CZP19004; and in part by the Youth Elite Project of State Ethnic Affairs Commission of China under Grant 2016-3-08. *(Corresponding author: Rongbo Zhu.)*

Yanhong Li and Rongbo Zhu are with the College of Computer Science, South Central University for Nationalities, Wuhan 430074, China (e-mail: liyanhong@mail.scuec.edu.cn; rbzhu@mail.scuec.edu.cn).

Shiwen Mao is with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849 USA (e-mail: smao@ieee.org).

Ashiq Anjum is with the College of Engineering and Technology, University of Derby, Derby, DE22 1GB, U.K. (e-mail: a.anjum@derby.ac.uk).

Digital Object Identifier 10.1109/JIOT.2020.2965730

As a kind of fog computing, vehicle fog computing (VFC) is considered as a promising method for supporting applications in IoV, which uses vehicles as an infrastructure to make full use of vehicle communication and computing resources. In particular, VFC utilizes a large number of cooperative end-user clients or near-user edge devices to perform huge amounts of communication and computation [3], which differs from other existing technologies in its proximity to end users, dense geographic distribution, and support for mobility [4], [5]. In order to enhance the computing and storage capabilities of the network edge, recently, a new network structure, named fog computing-based IoV (FC-IoV) [6], is proposed, which deploys fog servers at downtown intersections and accident-prone roads to enhance the computing and storage capabilities of the network edge.

Recently, lots of efforts are made to explore different kinds of issues on fog-based IoV, such as the optimal deployment and dimensionality (ODD) for autonomous driving [7] and reasonable and feasible resource allocation in real time [8]. However, there is few work on processing spatial–textual information generated in IoV to obtain user interested information. In real life, due to the popularity of on-board geographic devices, large numbers of spatial–textual objects are generated in IoV. To effectively process the massive data collected and obtain the information that users are interested in, spatial keyword query (SKQ) has been proposed and discussed [9]–[13], which uses a set of keywords and a spatial constraint to express user's interest in exploring useful information.

The existing work on SKQ query processing can be divided into two categories: 1) SKQ in Euclidean space [11] and 2) SKQ in traffic networks [14]. For SKQ in traffic networks, the schemes use real traffic network distance rather than the Euclidean distance in the Euclidean space, and thus can better meet the requirements of real-time applications in IoV. Moreover, considering that some previous work focuses on SKQ requiring exact keyword matching, and may result in too few results returned due to the diversity of textual expressions, recently, approximate SKQ (ASKQ) was explored. ASKQ can handle spelling errors and conversional spelling difference (for example, *color* versus *colour*), which appear in real applications frequently.

However, in many applications of IoV, such as mobile e-commence, various items are generated with textual descriptions, different attributes, and spatial locations.

Correspondingly, the requirement of a user could include a set of keywords, attribute-value pairs, distance limitation, or the number k of results, for example, "oxford," "dictionary," publish year = 2018 & price = 1000, and $k = 5$ (means the top-5 results). To capture the requirements of users, a spatial keyword search with numeric attributes is needed. Meanwhile, the more queries and objects involved, the more complex the query processing, which makes efficient query processing and fast feedback on query results a challenge. This calls for ASKQ (A^{2}SK) with locations, textual descriptions, and numeric attribute requirement simultaneously. To this end, we also need to make full use of the potential communication and computing power around query users in IoV, in addition to the efficient query processing methods.

To address the issues mentioned above, this article explores the fog computing-based A^{2}SK queries in traffic networks of IoV (A^{2}SKIV), which poses three major challenges. First, query users and textual–spatial objects may distribute within a large traffic networks with millions vertices and edges in IoV. How to efficiently calculate the network distances between queries and objects is the first issue need to be handled. Second, with millions of textual–spatial objects in IoV, we need to consider a large number of keywords and attribute-value pairs. Moreover, approximate keyword match rather than exact keyword match is considered which makes A^{2}SKIV search more complex. Third, many users may initiate queries simultaneously; the proposed matching method should be effective enough to significantly reduce the cost of query processing.

To support network distance pruning, keyword pruning, and numeric attribute-value pruning simultaneously, a novel spatial–textual hybrid index structure should be designed, which should consider the relative invariance of traffic network structure and the dynamic variation of textual–spatial objects and queries. First, we need a spatial index to keep the traffic network structure in IoV, thus given the positions of an object and a query, the network distance between them can be calculated quickly, while maintaining a reasonable and acceptable amount of storage space. Meanwhile, a textual and numeric index on the textual–spatial objects of each traffic network region (subgraph) is required too. In order to save space consumption, the textual information and numeric information need to be organized efficiently and smartly. Moreover, in order to improve the processing efficiency of a huge amount of unqualified textual–spatial objects, some efficient pruning rules are also needed.

In order to meet the requirements mentioned above, this article explores A^{2}SKIV comprehensively, and the main contributions of this article are as follows.

1) The A^{2}SKIV problem is formulated, which distinguishes itself from existing SKQ query efforts in that it takes into account textual similarity, numeric similarity, and spatial proximity in traffic network space, simultaneously.
2) A two-level spatial–textual hybrid index STAG-tree is presented. In addition, several lemmas are presented to prune a huge amount of unrelated objects. A top-k A^{2}SKIV query processing algorithm based on the STAG-tree index is designed. In addition, we discuss how to extend the proposed method for supporting numeric attributes with interval values.
3) Simulation using two traffic networks together with their spatial–textual object sets is performed to evaluate the effectiveness of the proposed STAG-tree index and query processing algorithm.

The remainder of this article is organized as follows. In Section II, we review the related work. Section III presents the system model and problem definitions. In Section IV, we introduce a hybrid index in detail. Top-k A^{2}SKIV query processing algorithm is proposed in Section V. Section VI discusses extending the method for supporting attributes with interval values. Section VII gives the experimental evaluation and, finally, Section VIII concludes this article.

II. Related Work

A. Fog Computing in IoV

In 2012, Cisco came up with the concept of fog computing. Since then, many efficient schemes were proposed [15]–[20]. An object cloud communication architecture [3] based on fog computing and intelligent gateway was proposed. Later, Aazam and Huh [21] proposed a system called fog micro data center, where the fog plays an important role in resource management, data filtering, preprocessing, data processing, and security measures. Meanwhile, Hou *et al.* [22] proposed a new concept of VFC, using vehicles as infrastructure to take full advantage of their communication and computation resources. An intelligent VFC system combining parking assistance and intelligent parking was discussed [7]. In particular, a vehicle reservation auction method based on VFC perception was designed to guide the vehicle to the available parking space with less effort during driving. Meanwhile, the vehicle's fog ability was utilized to compensate the vehicle's service cost through monetary reward, thus helping to delay the sensitive computing service. Yu *et al.* [6] discussed the ODD of FC-IoV infrastructure for autonomous driving. Two different architectural patterns, namely, coupling pattern and decoupling pattern, were proposed, and the ODD problem was transformed into two integer linear programming formulas to reduce the deployment cost. Such efforts improve the computing and storage capabilities of IoV and enable lots of applications. In edge-enabled networks, the geographic diversity of resources and various hardware configurations need to be carefully managed to ensure efficient utilization of resources. Lamb and Agrawal [8] analyzed the moving edge calculation of vehicle networks and introduced an architecture of evaluating available resources and allocating the most reasonable and feasible resources in real time.

B. SKQ Querying in Traffic Networks of IoV

In order to meet user's interests in IoV, lots of efforts are made to deal with moving top-k SKQ processing, direction-aware SKQ processing, interactive top-k SKQ querying, keyword search based on distributed graphs [23], why-not range-based skyline queries [24], and location-aware error-tolerant keyword search [25]. In order to accelerate the calculation of long road network distance, a multihop distance labeling scheme (DBM) was proposed [26], which is based on the Dijkstra method. Guo *et al.* [14] discussed the distributed SKQ search on the traffic network and proposed a new distributed index. By using this index, each machine independently evaluates search operations in a distributed manner. Gao *et al.* [27] discussed reverse top-k Boolean SKQ search in traffic networks, which shows how to use arbitrary k to answer the query without anticipatory computing. Zhao *et al.* [28] explored the time-aware SkQ queries on the traffic network. They proposed a novel TG index and several algorithms to efficiently process this type of queries. To support mobile search and targeted location-aware advertising, an inverted index-based solution (ILM) is proposed to improve query performance [29]. Li *et al.* [30] studied the intelligent augmented keyword search in real-life IoVs. A hybrid index called ASKTI was proposed. In ASKTI, the information of traffic network structure, keywords, Boolean expressions, and spatial information of objects are smartly organized, so as to prune unqualified traffic network space as early as possible. Abeywickrama *et al.* [31] discussed how to efficiently process SKQ queries on traffic networks and proposed K-SPIN, a versatile framework that avoids keyword separated indexes to reduce latency and avoid expensive operations.

To improve query processing performance in IoV, there are many similarity functions, such as edit distance, Jaccard, and n-gram [32]. To handle the inconsistencies and errors in queries and data, Alsubaiee *et al.* [33] proposed a natural index structure, which enhances the approximate keyword search ability of the spatial index based on tree. An approximate n-gram matching method was proposed [34], which uses the long but approximate n-gram matching as the basis for pruning k nearest neighbor candidates. Zheng *et al.* [35] explored approximate keyword search in semantic track database and proposed a hybrid index called Giki. Giki consists of two components, which are SQ-Tree part using n-grams and K-Ref part using edit distance.

Although there are many effective query processing methods in IoV, most schemes face the following limitations.

1) Focus on exact keyword match, while ignoring approximate keyword match, which can handle spelling errors and traditional spelling differences that often occur in practical applications.
2) Only keyword matching and attribute-value matching are considered, ignoring spatial constraints.
3) Limited to Euclidean space, and the query search cannot be processed in traffic networks in IoV, a realistic application scenario.

This article fills this gap by developing a two-level spatial–textual hybrid index, which can overcome the limitations mentioned above in IoV.

III. System Model and Problem Definitions

This section first gives the system model, and then formulates top-k A^{2}SKIV queries. Table I lists the notations that we use in this article.

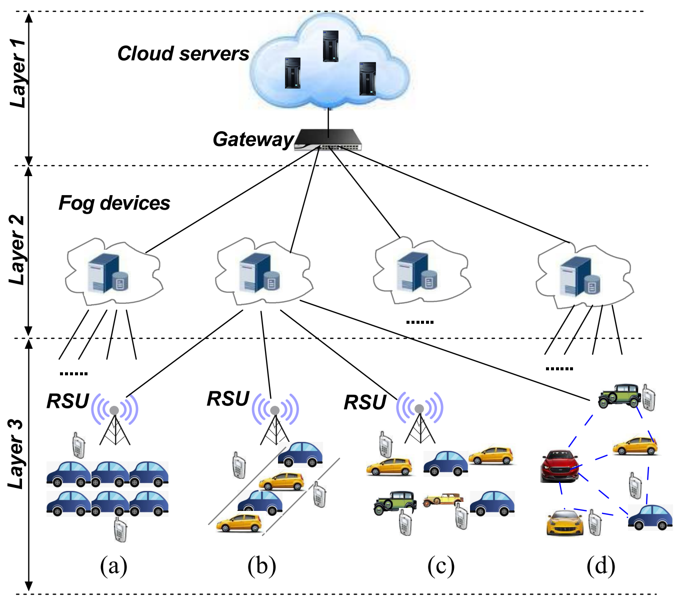


Fig. 1. Illustration of the system model. (a) Parking IoT. (b) Roadside parking. (c) JamCloud. (d) VANETs.

A. System Model

To meet the requirements of efficient query processing and fast feedback on query results, a fog computing-based network structure FCV is adopted to utilize the computing and storage capabilities of edge devices, which is a hierarchical structure that consists of three layers. Fig. 1 illustrates the system overview and scenarios of FCV with moving and parked vehicles' service and applications.

The proposed FCV considers four scenarios of vehicle behavior states. Fog computing has the natural advantage of being closer to vehicle endpoints and mobile devices, thus avoiding the high latency associated with complex system responses and service failures associated with remote routing to remote cloud servers. To address communication and computing power issues, FCV employs vehicles and mobile devices as the infrastructures, making full use of their communication and computing resources. Moreover, RSUs and fog devices are adopted and deployed. In general, the deployment of RSUs and fog devices focuses on intersections in the city center and some road-sides on busy roads.

As shown in Fig. 1, the first layer of FCV is a cloud computing layer which includes cloud servers and gateways. In particular, the gateway communicates with other heterogeneous networks and can also send the filtered underlying data to cloud servers.

The second one is the fog computing layer including lightweight fog devices at network edges. Fog devices temporarily cache and process the raw-date a collected, and upload the filtered data to the cloud servers for further processing. Fog devices can also store some frequently accessed data for rapid-response processing.

The third layer is the accessing layer, which includes RSUs, vehicles, and mobile devices. RSUs provide open service access points for fog computing vehicles and mobile devices. Note that although RSUs and fog devices are deployed in

TABLE I
SYMBOLS AND DEFINITIONS

Notation	Definition
o	A textual-spatial object
q	An A^{2}SKIV query
$q.V(o.V)$	A set of attribute-value pairs for q (o)
$q.L(o.L)$	The location of q (o)
$D_{td}(q,o)$	The textual distance between o and q
$D_{nd}(q,o)$	The numeric distance between o and q
$D_{tr}(q,o)$	The travel distance between o and q
$D_{tns}(q,o)$	The textual-numeric-spatial distance between o and q
$D^{LB}_{tns}(q,\mathcal{G}_i)$	The lower-bound textual-numeric-spatial distance between subgraph $\mathcal{G}_i$ and q
$d^{LB}_{ed}(q_i,w_i)$	The lower-bound edit distance between word w_i and query keyword q_i

similar locations, we will deploy them separately, taking into account the flexibility of deployment. RSUs, nearby vehicles, and mobile devices communicate wirelessly, exchanging information, and collaborating on computing tasks. However, RSUs communicate with fog devices via wired connection. There are four types of scenarios in the third layer, as described below.

Fig. 1(a) and (b) illustrates the parked vehicles and mobile devices as infrastructures. A huge number of parked vehicles are scattered across the traffic network in IoV. These vehicles and mobile devices become a rich computing infrastructure, providing powerful computing resources and storage space. When joining the FCV, they can be used as a small data center to deal with a variety of complex tasks. Fig. 1(c) and (d) illustrates moving vehicles working as infrastructures. In urban areas, traffic is usually slow. In addition, most vehicles travel very slowly when entering the urban area, especially during rush hours, and there is a good communication between nearby mobile vehicles and devices. Moving vehicles can constantly transmit information by establishing new connections. When nearby moving vehicles join the FCV, they can collaborate and connect with each other and complete tasks using local computing and communication resources.

B. Problem Definition

1) Traffic Network of IoV: A traffic network of IoV is modeled as a undirected weighted graph $G = (V, E)$, where V is a set of vertices, and E is a set of edges. A vertice $v \in V$ represents a road intersection or endpoint in the traffic network. An edge $e(v_i, v_j, l) \in E$, represents the road segment between two vertices v_i and v_j $(i \neq j)$, and l represents the length of the road segment. Our model can be extended to support the directed weighted graph, which represents unidirectional traffic by simply allowing the length of $e(v_i, v_j)$ be set different from that of $e(v_j, v_i)$.

2) Spatial–Textual Objects With Numeric Attributes and Approximate Spatial Keyword Queries:

Definition 1 [Spatial–Textual Objects With Numeric Attributes in Traffic Networks of IoV (Object for Short)]: Object o is defined as $o = (o.\text{tags}, o \cdot V, o \cdot L)$, where o.tags is related descriptive tags containing a set of keywords, $o \cdot V$ is a set of attribute-value pairs, and $o \cdot L$ is a spatial point on the edge of the traffic network. The size of $o \cdot V$ is the number of attribute-value pairs represented by n, and so o can be represented as

$$o = \left\{\text{tags}, A_1 = v_1 \bigcap A_2 = v_2 \bigcap \cdots \bigcap A_n = v_n,\ o \cdot L\right\}.$$

Definition 2 [Approximate Spatial Keyword Queries With Numeric Attributes in IoV (A^{2}SKIV)]: An A^{2}SKIV query q is defined as $q = (q \cdot W, q \cdot V, q \cdot L)$, where $q \cdot W$ is the relevant keywords, $q \cdot V$ is a set of user-given attribute-value pairs, and $q \cdot L$ is a spatial point on the edge of the traffic network. The size of $q \cdot V$ is the number of attribute-value pairs represented by m, and so q can be represented as

$$q = \left\{q \cdot W, A_1 = v_1 \bigcap A_2 = v_2 \bigcap \cdots \bigcap A_m = v_m,\ q \cdot L\right\}.$$

3) Match Semantics: For A^{2}SKIV query q and object o, to measure the relevance between q and o, there are three aspects should be considered, i.e., textual distance, numeric attribute distance, and traffic network distance between q and o.

Definition 3 (Keyword Mapping): For A^{2}SKIV query q and object o, a keyword mapping from q to o, i.e., q.KM(o), is a set of keywords, in which each keyword is textual closest to q among all keywords contained by o in terms of edit distance,[1] i.e., $w_i = \arg\min_{w_j \in o.\text{tags}}\{d_{\text{ed}}(q_i, w_j)\}$.

Definition 4 (Textual Distance): Given A^{2}SKIV query q and object o, we first calculate the sum of edit distance between each keyword $w_i \in q.KM(o)$ and corresponding keyword $q_i \in q \cdot W$. To normalize the sum of edit distance calculated to range [0, 1], the $\max\{|q \cdot W|, |o.\text{tags}|\}$, which is the greater one between $|q \cdot W|$ and $|o.\text{tags}|$, is also considered as follows:

$$D_{\text{td}}(q, o) = \sum_{q_i \in q \cdot W} \frac{d_{\text{ed}}(q_i, w_i)}{|q \cdot W| \times \max\{|q \cdot W|, |o.\text{tags}|\}}. \quad (1)$$

Next, let us discuss how to calculate the numeric distance between query q and object o. Numeric attribute distance refers to the degree of difference between the values of q and o under the same numeric attribute, which is expressed as the size of difference.

For q and o, the numeric distance between q and o under each numeric attribute A_j $(1 \leq j \leq m)$ can be expressed as follows:

$$d_j = \begin{cases} d\big(q.A_j, o.A_j\big), & \text{if } o.A_j \text{ exists} \\ +\infty. & \text{otherwise.} \end{cases} \quad (2)$$

Then, we normalize each numeric attribute distance to range [0, 1], and comprehensively consider the influence of each numeric attribute distance to calculate the total numeric distance between q and o.

Definition 5 (Numeric Distance): For each query attribute $A_j \in q \cdot V$, let $M_j = \text{Max}(A_j) - \text{Min}(A_j) = \beta_j \times 10^{c^j}$, where

[1]The edit distance between the two strings s_1 and s_2, $d_{\text{ed}}(s_1, s_2)$, can be defined as the minimum number of edit operations (i.e., insertion, deletion, or substitution), required to convert from s_1 to s_2. The n-gram is a common technique for estimating edit distance between strings. For a string s, the n-grams can be obtained by sliding a window of length n from the beginning to the end of the string. In particular, the Minhash method [36] can be used to estimate set similarity.

$\text{Max}(A_j)$ and $\text{Min}(A_j)$ are the maximum and minimum values of attribute A_j for all objects in object set O, and $1.0 \leq \beta_j \leq 10.0$. Let $e_j = c_j + 1 \geq 1$, the numeric distance $D_{\text{nd}}(q, o)$ between q and o can be defined as follows:

$$D_{\text{nd}}(q, o) = \frac{1}{|q \cdot V|} \sum_{A_j \in q \cdot V} \left(\frac{d_j}{M_j} \right)^{\frac{1}{e_j}}. \tag{3}$$

Note: If there is any query attribute that is not in $o \cdot V$, $D_{\text{nd}}(q, o) = +\infty$.

Travel distance is another aspect for query effort measurement, which is the length of the shortest path from query q to object o, i.e., $D_N(q.l, o.l)$.

Definition 6 (Travel Distance): Since the value of the Sigmoid function changes rapidly in the case of small variables, this is consistent with the intuition that user satisfaction is generally more sensitive to travel distance in the case of short distance. Therefore, we use the Sigmoid function to normalize travel distance to range [0, 1]

$$D_{\text{tr}}(q, o) = \frac{2}{1 + e^{-\rho \times D_N(q.l, o.l)}} - 1 \tag{4}$$

where $0 < \rho \leq 1$ is the distance adjustment parameter.

Finally, we adopt the concept of textual–numeric–spatial distance and combine the measurement of spatial, textual, and numeric relevance between q and o by using a simple linear interpolation. In particular, the textual–numeric–spatial distance between q and o is a linear combination of the spatial, textual, and numeric relevance between q and o, each weighted with parameters α, β, and γ, respectively.

Definition 7 (Textual–Numeric–Spatial Distance): Formally, given q and o, the textual–numeric–spatial distance is denoted as $D_{\text{tns}}(q, o)$, which is defined as

$$D_{\text{tns}}(q, o) = \alpha \times D_{\text{td}}(q, o) + \beta \times D_{\text{nd}}(q, o) + \gamma \times D_{\text{tr}}(q, o) \tag{5}$$

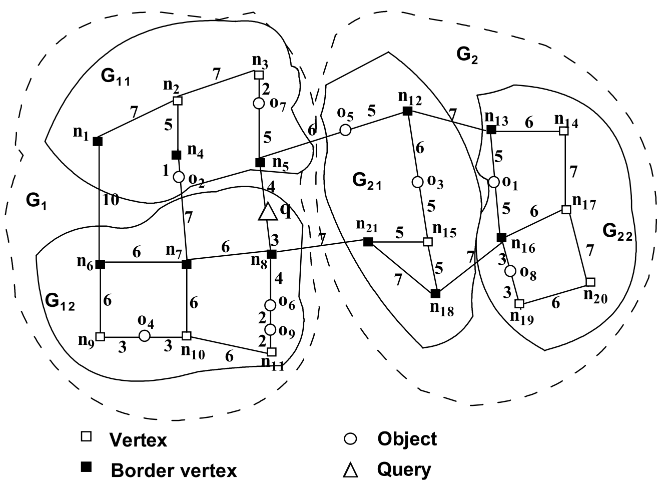
where $\alpha, \beta, \gamma \geq 0$, and $\alpha + \beta + \gamma = 1$.

C. Problem Statement

By using the textual–numeric–spatial distance $D_{\text{tns}}(q, o)$ to measure the combined proximity between query q and object o, we can formally define top-k A^{2}SKIV query below.

Definition 8 (Top-k A^{2}SKIV Query): Given a spatial–textual object data set O, a Top-k A^{2}SKIV query $q = (q \cdot W, q \cdot V, q \cdot L, k)$ retrieves a set of objects $\widehat{O} \subseteq O$, such that $|\widehat{O}| = k$ and $\forall o \in \widehat{O}$ and $o' \in O - \widehat{O}$, $D_{\text{tns}}(q, o) < D_{\text{tns}}(q, o')$.

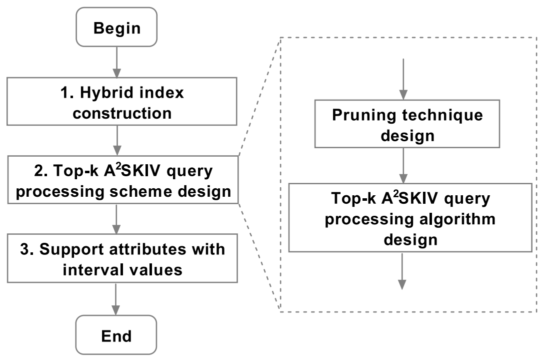
Example 1: Fig. 2 illustrates an example of Top-k A^{2}SKIV query on the traffic network in IoV, with ten spatial–textual objects and one query located on the edges. Each object has a set of keywords and a set of attribute-value pairs to provide its description information, and a spatial point on the edge of the traffic network to describe its location. The query q contains four items: 1) a set of query keywords {Theater, coffee}; 2) attribute-value pairs for "$A_1 = 4.4$ & $A_2 = 45$"; 3) a spatial point q_l for its current location; and 4) a value $k = 1$ for top-1 related objects wanted. Note that A_1 ="rating," and A_2 = "pcc (per capita consumption)." We first consider o_5, o_6, o_7, and o_9, whose network distances from query q are the four most shortest ones among all the objects in O.



Textual-spatial obejcts:
o_1:({Restaurant,Theater, Mall},{A_1=4.5,A_2=70,A_3=yes},l_1)
o_2:({Bakery,Pizza,Bread},{A_1=4.6,A_2=65,A_3=yes},l_2)
o_3:({Coffee,Italian},A_1=4.2,A_2=78},l_3)
o_4:({Coffee,bread},{A_1=4.6,A_2=55,A_3=yes},l_4)
o_5:({Theater,Italian,Coffee},{A_1=4.6,A_2=50,A_3=yes},l_5)
o_6:({Theatre,Coffee,Bread},{A_1=4.3,A_2=45,A_3=yes},l_6)
o_7:({Italian,Fish,Bread},{A_1=4.6,A_2=62,A_3=no},l_7)
o_8:({Theatre,Pizza,Coffee},{A_1=4.5,A_2=56,A_3=yes},l_8)
o_9:({Theatre,Coffee},{A_1=4.8,A_3=yes},l_8)

A_1: rating; A_2: pcc; A_3: on_smoking;

Fig. 2. Example of Top-k A^{2}SKIV query.



Begin
1. Hybrid index construction
2. Top-k A²SKIV query processing scheme design
3. Support attributes with interval values
End
Pruning technique design
Top-k A²SKIV query processing algorithm design

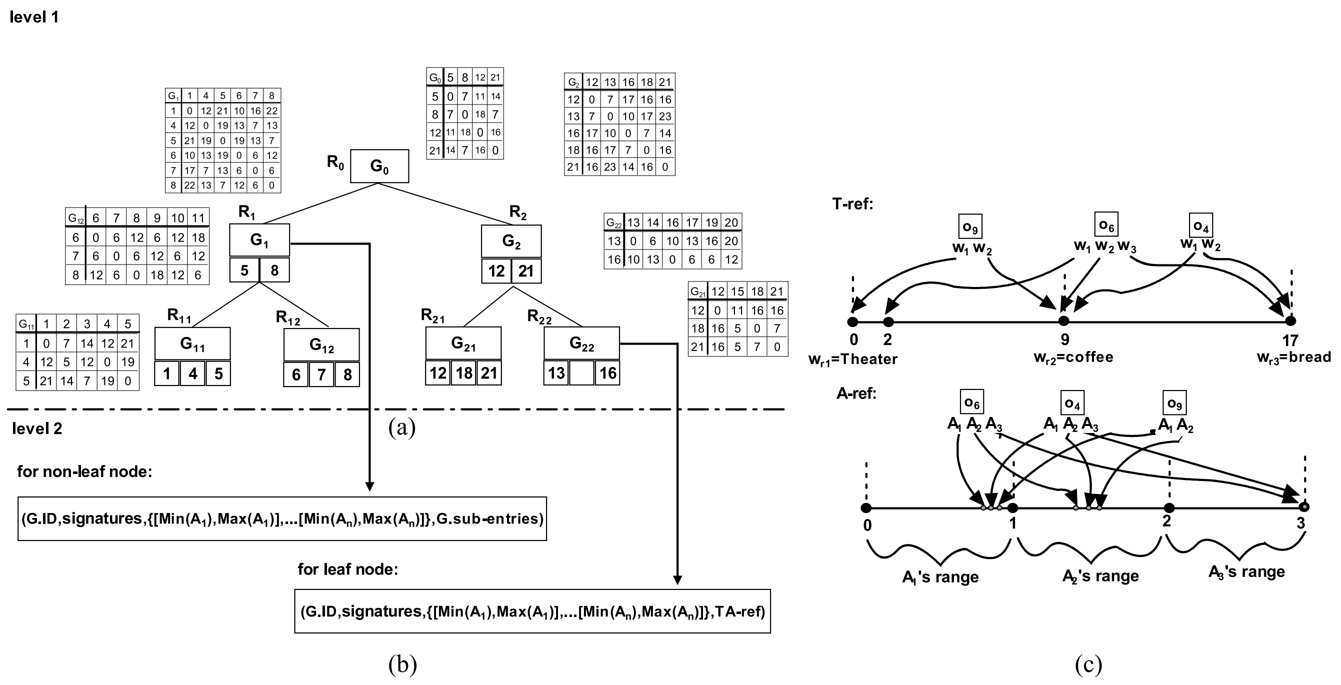
Fig. 3. Flowchart of Top-k A^{2}SKIV query processing.

Assume $M_1 = \text{Max}\ (A_1) - \text{Min}\ (A_1) = 5 - 0 = 5$, $M_2 = \text{Max}\ (A_2) - \text{Min}\ (A_2) = 200 - 0 = 200$, $e_1 = 1$, $e_2 = 3$, $\rho = 0.1$, and $\alpha = \beta = \gamma = (1/3)$, we can get

$D_{\text{tns}}(q, o_6) = (1/3) \times (1/2) \times ([1/7] + 0) + (1/3) \times (1/2) \times ([0.1/5] + 0) + (1/3) \times ([2/(e^{-0.1*7})] - 1) = 0.1631$.

Similarly, we can get $D_{\text{tns}}(q, o_5) = 0.1649, D_{\text{tns}}(q, o_7) = 0.1682$, and $D_{\text{tns}}(q, o_9) = +\infty$. Note that $D_{\text{nd}}(q, o_9) = +\infty$ since o_9 does not have query attribute A_2, thus $D_{\text{tns}}(q, o_9)$ equals $+\infty$. Then, object o_6 is the top-1 result object of q at this moment, and other objects can be evaluated similarly.

In the following three sections, the detailed method for top-k A^{2}SKIV query processing is proposed, which includes hybrid index construction, Top-k A^{2}SKIV processing scheme design, and extending our index constructed to support attributes with interval values. Top-k A^{2}SKIV processing scheme consists of pruning techniques and query processing algorithms. The query processing flowchart is then shown in Fig. 3.



G_0	5	8	12	21
5	0	7	11	14
8	7	0	18	7
12	11	18	0	16
21	14	7	16	0

G_1	1	4	5	6	7	8
1	0	12	21	10	16	22
4	12	0	19	13	7	13
5	21	19	0	19	13	7
6	10	13	19	0	6	12
7	17	7	13	6	0	6
8	22	13	7	12	6	0

G_2	12	13	16	18	21
12	0	7	17	16	16
13	7	0	10	17	23
16	17	10	0	7	14
18	16	17	7	0	16
21	16	23	14	16	0

G_{12}	6	7	8	9	10	11
6	0	6	12	6	12	18
7	6	0	6	12	6	12
8	12	6	0	18	12	6

G_{22}	13	14	16	17	19	20
13	0	6	10	13	16	20
16	10	13	0	6	6	12

G_{11}	1	2	3	4	5
1	0	7	14	12	21
4	12	5	12	0	19
5	21	14	7	19	0

G_{21}	12	15	18	21
12	0	11	16	16
18	16	5	0	7
21	16	5	7	0

Fig. 4. STAG-tree index. (a) G-tree component. (b) Textual and numeric component. (c) TA-ref index for leaf subgraph.

IV. Hybrid Index for A^{2}SKIV Query Processing

To improve query performance and efficiently prune irrelevant objects for A^{2}SKIV queries as many as possible, a novel two-level spatial–textual hybrid index structure STAG-tree is proposed as shown in Fig. 4, which supports network distance pruning, textual pruning, and numeric attribute pruning simultaneously. STAG-tree also considers the relative invariance of traffic network structure and the dynamic variation of objects and queries. Then, the flowchart of building the STAG-tree is illustrated in Fig. 5.

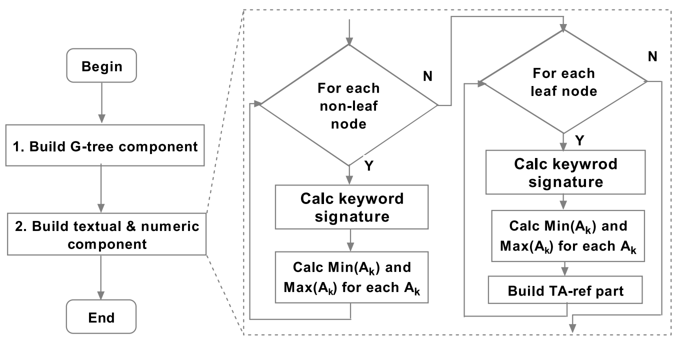

Begin
1. Build G-tree component
2. Build textual & numeric component
End
For each non-leaf node
N
Y
Calc keyword signature
Calc Min(A$_k$) and Max(A$_k$) for each A$_k$
For each leaf node
N
Y
Calc keywrod signature
Calc Min(A$_k$) and Max(A$_k$) for each A$_k$
Build TA-ref part

Fig. 5. Flowchart of building the STAG-tree index.

A. Build G-Tree Component

G-tree [37] is an assembly based index and can efficiently support location-based queries on traffic network in IoV. A traffic network is modeled by an undirected weighted graph $\mathcal{G} = \{V, E\}$ as mentioned before, and G-tree can be constructed by using graph partitioning. First, the graph $\mathcal{G}$ is marked as the root of G-tree, and then $\mathcal{G}$ is partitioned into f equal-sized subgraphs $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_f$, i.e., $|V_{\mathcal{G}_1}|, |V_{\mathcal{G}_2}|, \ldots, |V_{\mathcal{G}_f}|$ are almost the same, and works as the parent node of these subgraphs. Note for $\mathcal{G}_i$ may exist $u \in V_i$ such that $\exists(u, v) \in E$ and $v \notin V_i$, such node u is called a border, and $B_{\mathcal{G}_i}$ is used to represent the border set in graph $\mathcal{G}_i$. Thus, $\mathcal{G}_i$ can be denoted by $\mathcal{G}_i = \{E_{\mathcal{G}_i}, V_{\mathcal{G}_i}, B_{\mathcal{G}_i}\}$, where $E_{\mathcal{G}_i}, V_{\mathcal{G}_i}$, and $B_{\mathcal{G}_i}$ denote the vertices, edges, and borders in $\mathcal{G}_i$ which meet the following conditions: 1) $\bigcup_{1 \geq i \geq f} V_{\mathcal{G}_i} = V$; 2) for $i \neq j$, $V_{\mathcal{G}_i} \bigcap V_{\mathcal{G}_j} = \emptyset$; 3) for $\forall u, v \in V_{\mathcal{G}_i}$, if $(u, v) \in E_{\mathcal{G}_i}$, then $(v, u) \in E_{\mathcal{G}_i}$; and 4) for $\forall u \in V_{\mathcal{G}_i}, \exists(u, v) \in E$ and $v \notin V_i$, then $u \in B_{\mathcal{G}_i}\}$. Then, subgraph $\mathcal{G}_i$ is partitioned recursively, and the steps are repeated until each subgraph has no more than τ vertices. Note that f and τ are adjustable parameters. For example, as shown in Fig. 2, the traffic network $\mathcal{G}_0$ is first divided into two subgraphs $\mathcal{G}_1$ and $\mathcal{G}_2$. Then, $\mathcal{G}_1$($\mathcal{G}_2$, resp.) $\mathcal{G}_{11}$ and $\mathcal{G}_{12}$ ($\mathcal{G}_{21}$ and $\mathcal{G}_{22}$, resp.). Assume $f = 2$ and $\tau = 6$, the G-tree structure of the traffic network in Fig. 2 can be obtained as shown in Fig. 4(a). Note that the numbers under the ID of each subgraph are the IDs of its borders.

To accelerate the shortest path calculation, G-tree keeps the distance metrics (DM) which include the shortest-path distance between each border–border pair (border–vertex pair, resp.) for nonleaf nodes (leaf nodes, resp.). Particularly, an efficient bottom-up method is adopted to accelerate the distance computation. In this way, the DMs of the G-tree in Fig. 4(a) can be obtained, and the DM of each subgraph (or graph) is given next to it. The total space complexity of G-tree is $O(\log_2 f * \sqrt{\tau} * |V| + \log_f(|V|/\tau) * \log_2^2 f * |V|)$, where $|V|$ is total number of vertices in graph $\mathcal{G}$, f is the fan-out of nonleaf G-tree nodes for graph $\mathcal{G}$, and τ is the maximum number of vertices contained in each leaf node of

G-tree. Note that $\log_2^2$, $\sqrt{\tau}$, and $\log_f(|V|/\tau)$ are small numbers, thus the size of G-tree is scalable. Please refer to [37] for details.

B. Build Textual and Numeric Component

Second, as shown in Fig. 4(b), the dynamic part of the index, i.e., a textual and numeric index on objects, is constructed. For each nonleaf subgraph (node) $\mathcal{G}_i$:

1) ID of the subgraph $\mathcal{G}_i$ is stored;
2) calculate and keep: a) the keyword signature of all the objects within $\mathcal{G}_i$ and b) the Min(A_k) and Max(A_k) of each numeric attribute A_k, which are the minimum and maximum values of A_k for all the objects in $\mathcal{G}_i$. If no object in $\mathcal{G}_i$ has attribute A_k, $\text{Max}(A_k) = \text{Min}(A_k) = +\infty$;
3) the entries pointing to the subgraphs of $\mathcal{G}_i$ is stored.

For each leaf subgraph (node) $\mathcal{G}_i$, we also calculate and keep the first two items similar to the nonleaf subgraph:

1) ID of $\mathcal{G}_i$;
2) the keyword signature, and Min(A_k) and Max(A_k) for each numeric attribute A_k.

The third item of the nonleaf subgraph is not required in the leaf subgraph, since it does not have any subgraphs. In addition, for each leaf subgraph, we also construct and keep the TA-ref index part as follows.

TA-Ref Index: TA-ref index, as shown in Fig. 4(c), is used to organize the textual and numerical information of the objects in each nonleaf subgraph, to facilitate textual distance and numeric distance calculation of objects in subgraphs. TA-ref index consists of two parts: 1) T-ref and 2) A-ref.

T-Ref Part: As far as we know, it is unfeasible to calculate the edit distance during query processing by directly using the Wagner–Fischer algorithm [32]. Thus, for each leaf subgraph $\mathcal{G}_i$, we construct the T-ref part to index the edit distance of the objects within $\mathcal{G}_i$. For $\mathcal{G}_i$, we select a set of reference keywords $R(\mathcal{G}_i)$= $\{w_r^{\mathcal{G}_i}\}$ to index the edit distances between the keywords contained in the objects within $\mathcal{G}_i$ and $R(\mathcal{G}_i)$.

To construct the T-ref part for $\mathcal{G}_i$, we need to divide the keywords contained in all the objects within $\mathcal{G}_i$ into N clusters, and select a reference keyword $w_{r_n}^{\mathcal{G}_i}$ for each cluster, thus to minimize the mathematical expectation of editing distance in each cluster. To this end, k-means clustering algorithm is adopted to obtain each cluster and its corresponding reference keyword. Thus, each object o_i within $\mathcal{G}_i$ is indexed in a B$^+$-tree by the key $\text{y}(o_i^n)$. The key $\text{y}(o_i^n)$ is calculated according to the edit distance between the keyword w_i^j and the reference keyword $w_{r_n}^{\mathcal{G}_i}$, i.e., $\text{y}(o_i^n) = d_{\text{ed}}(w_{r_n}^{\mathcal{G}_i}, w_i^j) + n \times C(0 \le n \le N)$, where C equals the maximum edit distance between the reference keyword of the cluster and the keywords belonging to the cluster. To facilitate the edit distance calculation in query processing, we also calculate and keep the distance lower limit $\text{DL}(w_{r_n}^{\mathcal{G}_i})$ and the distance upper limit $\text{DU}(w_{r_n}^{\mathcal{G}_i})$ for each cluster.

Example 2: Fig. 4(c) gives the T-ref for subgraph $\mathcal{G}_{12}$, where the keywords of objects within $\mathcal{G}_{12}$ are partitioned into three clusters, whose reference keyword is "Theater," "coffee," and "bread," respectively.

A-Ref Part: A-ref part is to facilitate the numeric distance calculation of the objects in subgraphs. For each numeric attribute A_k $(1 \le k \le n)$ of the system, we use $[k-1, k)$ to represent the value range of the objects with attribute A_k. To map the attribute values of objects to the value ranges of attributes, each object o_i within $\mathcal{G}_i$ is indexed in a B$^+$-tree by the key $\text{y}(o_i^k)$. The key $\text{y}(o_i^k)$ is calculated according to its attribute value, i.e., $\text{y}(o_i^k) = ([o_i.V_k]/[M_k]) + k - 1(0 \le k \le n)$, where $o_i.V_k$ is the attribute value of object o_i for A_k, and $M_k = \text{Max}(A_k) - \text{Min}(A_k)$. Note that Max($A_k$) and Min($A_k$) are the maximum and minimum values of attribute A_k for all the objects in O.

Example 3: Fig. 4(c) also gives the A-ref for subgraph $\mathcal{G}_{12}$, where the numeric attributes of objects within $\mathcal{G}_{12}$ are partitioned into three clusters, whose value range is [0, 1), [1, 2), and [2, 3), respectively. For example, the attribute value for A_2 of o_6 is 45, and $M_2 = 100$, and then we have $\text{y}(o_6^2) = (45/100) + 2 - 1 = 1.45$.

Remember, we partition the traffic network into equally sized subgraphs, while minimizing the number of border vertices at the same time. And then, the index part of each subgraph is constructed accordingly. To allocate the workload among different fog-devices in the second layer of our FCV structure, the information of STAG-index is partitioned, each corresponding to a subgraph. For a fog server, the index part of the subgraph on which it resides and the subgraphs surrounding it will be stored in the server.

V. Process A^{2}SKIV Queries in IoV

This section introduces the top-k A^{2}SKIV query processing method based on the STAG-tree index.

A. Pruning Techniques

First, several lemmas are introduced to efficiently prune the unrelated traffic network space and unqualified spatial–textual objects in IoV.

Lemma 1: Given a top-k A^{2}SKIV query $q = (q \cdot W, q \cdot V, q \cdot L, k)$ and a subgraph $\mathcal{G}_i$, $\mathcal{G}_i$ can be ignored if

$$d_N^{\min}(\mathcal{G}_i, q) > -\frac{1}{\rho}\ln\left(\frac{2}{\frac{D_{\text{tns}}(q,o_k)}{\gamma}+1} - 1\right)$$

where o_k is the kth nearest neighbor of q.

Proof: For any object o in $\mathcal{G}_i$, we have

$$\begin{aligned} D_{\text{tns}}(q,o) &= \alpha \times D_{\text{td}}(q,o) + \beta \times D_{\text{nd}}(q,o) + \gamma \times D_{\text{tr}}(q,o) \\ &\ge \gamma \times D_{\text{tr}}(q,o) \end{aligned}$$

$$\text{if}\quad d_N^{\min}(\mathcal{G}_i, q) > -\frac{1}{\rho}\ln\left(\frac{2}{\frac{D_{\text{tns}}(q,o_k)}{\gamma}+1} - 1\right)$$

through transformation, for each $o \in \mathcal{G}_i$, we have

$$D_{\text{tr}}(q,o) > \frac{D_{\text{tns}}(q,o_k)}{\gamma}.$$

Thus, we have

$$D_{\text{tns}}(q,o) \ge \gamma \times D_{\text{tr}}(q,o) > D_{\text{tns}}(q,o_k).$$

As a result, any object o in $\mathcal{G}_i$ cannot be a top-k result object. Thus, $\mathcal{G}_i$ can be ignored. ■

Lemma 2: Given a top-k A^{2}SKIV query $q = (q \cdot W, q \cdot V, q \cdot L, k)$ and a subgraph $\mathcal{G}_i$, if $\forall q_j \in q \cdot W$, q_j.signature $\cap$ $\mathcal{G}_i$.signature $= \emptyset$, then $\mathcal{G}_i$ can be ignored.

Proof: For $\mathcal{G}_i$, if $\forall q_j \in q \cdot W$, q_j.signature$\cap\mathcal{G}_i$.signature $= \emptyset$, which means that for any query keyword q_j, there is no object in $\mathcal{G}_i$ textual similar with q_j, hence, $\mathcal{G}_i$ can be ignored. ■

Lemma 3: Given a top-k A^{2}SKIV query $q = (q \cdot W, q \cdot V, q \cdot L, k)$ and a subgraph $\mathcal{G}_i$, if $\exists A_j \in q \cdot V$, such that $\text{Max}(A_j)(or\text{Min}(A_j))$ for $\mathcal{G}_i$ equals $+\infty$, then $\mathcal{G}_i$ can be ignored.

Proof: For subgraph $\mathcal{G}_i$, if $\exists A_j \in q \cdot V$, whose $\text{Max}(A_j)$ (or $\text{Min}(A_j)$) for $\mathcal{G}_i$ equals $+\infty$, it means $\mathcal{G}_i$ does not include any object containing attribute A_j, which also means that the numeric distance d_j for any object o in $\mathcal{G}_i$ equals $+\infty$. Thus, $D_{\text{nd}}(q, o)$ equals $+\infty$, which in turn makes $D_{\text{tns}}(q, o)$ equal $+\infty$. Any object o in subgraph $\mathcal{G}_i$ cannot be a top-k result object. Thus, $\mathcal{G}_i$ can be ignored. ■

Lower Bound Distance Computation: The pruning strength of the above three lemmas is relatively limited. In order to further reduce unrelated subgraphs, for any subgraph $\mathcal{G}_i$, we calculate $D^{\text{LB}}_{\text{tns}}(q, \mathcal{G}_i)$ as follows.

1) $\mathcal{G}_i$ is nonleaf subgraph.
   a) If $\mathcal{G}_i$ is not pruned by Lemmas 1, 2, or 3, we reduce $D^{\text{LB}}_{\text{tns}}(q, \mathcal{G}_i)$ by assuming $D_{\text{td}}(q, \mathcal{G}_i)$ equals 0.
   b) To calculate $D^{\text{LB}}_{\text{nd}}(q, \mathcal{G}_i)$, for each query attribute A_k, we compare its value for A_k, i.e., $q.A_k.v$, with the value range $[\text{Min}(A_k), \text{Max}(A_k)]$ of $\mathcal{G}_i$. If $q.A_k.v$ falls in $[\text{Min}(A_k), \text{Max}(A_k)]$, assume that $d_k = 0$; otherwise, $d_k = \min\{|q.A_k.v - \text{Min}(A_k)|, |q.A_k.v - \text{Max}(A_k)|\}$. Then, $D^{\text{LB}}_{\text{nd}}(q, \mathcal{G}_i)$ is obtained by (3).
   c) To calculate $D^{\text{LB}}_{\text{tr}}(q, \mathcal{G}_i)$, we use the shortest network distance between q and the border vertices of $\mathcal{G}_i$, if q does not belong to $\mathcal{G}_i$; otherwise, $D^{\text{LB}}_{\text{tr}}(q, \mathcal{G}_i) = 0$.

   Finally, $D^{\text{LB}}_{\text{tns}}(q, \mathcal{G}_i) = \alpha \times 0 + \beta \times D^{\text{LB}}_{\text{nd}}(q, \mathcal{G}_i) + \gamma \times D^{\text{LB}}_{\text{tr}}(q, \mathcal{G}_i)$.
2) $\mathcal{G}_i$ is leaf subgraph.
   a) The calculation of $D^{\text{LB}}_{\text{nd}}(q, \mathcal{G}_i)$ and $D^{\text{LB}}_{\text{tr}}(q, \mathcal{G}_i)$ is the same as that of the nonleaf subgraph.
   b) The TA-ref index of leaf subgraph $\mathcal{G}_i$ will be used to calculate $D^{\text{LB}}_{\text{tns}}(q, \mathcal{G}_i)$, whose focus is the calculation of $d^{\text{LB}}_{\text{ed}}(q_i, w_i)$ for each query keyword $q_i \in q \cdot W$ and its most mapping keyword w_i for objects in $\mathcal{G}_i$. We will detail how to determine w_i and its corresponding object o_i as follows.

Calculating $d^{LB}_{ed}(q_i, w_i)$ for q_i: Since the edit distance follows the triangle inequality, we make use of the edit distances between q and reference keywords of the T-ref part in TA-ref index for $\mathcal{G}_i$.

First, the edit distance between q_i and each reference keyword $w^{\mathcal{G}_i}_{rj}$, i.e., $d_{\text{ed}}(q_i, w^{\mathcal{G}_i}_{rj})$, is calculated. If $d_{\text{ed}}(q_i, w^{\mathcal{G}_i}_{rj}) \in [\text{DL}(w^{\mathcal{G}_i}_{rj}), \text{DU}(w^{\mathcal{G}_i}_{rj})]$, let $d^{\text{LB}}_{\text{ed}}(q_i, w_i) = 0$, and the processing for k_i completes.

Otherwise, we choose $w_r = \arg\min_{0\leq j\leq n} \{d_{\text{ed}}(q_i, w^{\mathcal{G}_i}_{rj})\}$ and its two bounding values $\text{DL}(w^{\mathcal{G}_i}_{r})$ and $\text{DU}(w^{\mathcal{G}_i}_{r})$, and let $d^{\text{LB}}_{\text{ed}}(q_i, w_i)$ equal $\min((d_{\text{ed}}(q_i, w^{\mathcal{G}_i}_{r}) - \text{DL}(w^{\mathcal{G}_i}_{r})), d_{\text{ed}}(q_i, w^{\mathcal{G}_i}_{r}) - \text{DU}(w^{\mathcal{G}_i}_{r})))$. Then, by using all the $d^{\text{LB}}_{\text{ed}}(q_i, w_i)$, $D^{\text{LB}}_{\text{td}}(q, \mathcal{G}_i)$ can be obtained through (1).

Finally, $D^{\text{LB}}_{\text{tns}}(q, \mathcal{G}_i) = \alpha \times D^{\text{LB}}_{\text{td}}(q, \mathcal{G}_i) + \beta \times D^{\text{LB}}_{\text{nd}}(q, \mathcal{G}_i) + \gamma \times D^{\text{LB}}_{\text{tr}}(q, \mathcal{G}_i)$.

Lemma 4: Given a top-k A^{2}SKIV query $q = (q\cdot W, q\cdot V, q\cdot L, k)$ and a subgraph $\mathcal{G}_i$, if $D^{\text{LB}}_{\text{tns}}(q, \mathcal{G}_i) > D_{\text{tns}}(q, o_k)$, where o_k is with the same meaning of Lemma 1, and $D^{\text{LB}}_{\text{tns}}(q, \mathcal{G}_i)$ is the lower bound of the textual–numeric–spatial distance between query q and any object o in $\mathcal{G}_i$, $\mathcal{G}_i$ can be ignored.

Proof: Since $D^{\text{LB}}_{\text{tns}}(q, \mathcal{G}_i) > D_{\text{tns}}(q, o_k)$, for any object $o \in \mathcal{G}_i$, there exist at least k objects whose textual–numeric–spatial distance between query q is smaller than that of o, thus o cannot be a top-k result object. Hence, $\mathcal{G}_i$ can be ignored. ■

B. *A^{2}SKIV Query Processing Algorithm*

Now we are ready to discuss the A^{2}SKIV query processing algorithm using STAG-tree index, which is called A^{2}S^{2}KG. It takes as inputs an STAG-tree $\mathcal{ST}$ and an A^{2}SKIV query $q = (q \cdot W, q \cdot V, q \cdot L, k)$, and outputs the result object set S_{result}. A^{2}S^{2}KG progressively accesses the nearest subgraphs and retrieves the most relevant objects. Finally, the k objects with the smallest textual–numeric–spatial distance value, $D_{\text{tns}}(q, o)$, form the query result set.

The detailed steps of the A^{2}S^{2}KG algorithm are shown in Algorithm 1. First, a min-heap HG is initialized to empty for organizing the nodes (subgraphs) or objects to be visited. Moreover, a set S_{result} is adopted to keep the result objects for query q, and a float D_{tsk} is initialized to be $+\infty$ for keeping the textual–numeric–spatial distance of the current kth nearest neighbor from query q. In particular, HG is an ordered structure and $D^{\text{LB}}_{\text{tns}}(q, P_{\text{node}})$ is the key of a node (subgraph) P_{node} in HG.

A^{2}S^{2}KG first locates the leaf node (subgraph), leaf(q), where q lies in. For each object o in leaf(q), it inserts o together with its $D_{\text{tns}}(q, o)$ into heap HG, and updates D_{tsk} accordingly, if $D_{\text{tns}}(q, o)$ is no larger than D_{tsk} (lines 4–6). Then, it uses pointer P_{node} to keep the upper most node (subgraph) visited of $\mathcal{ST}$ and uses variable P_{LB} to keep the lower bound of the textual–numeric–spatial distance between query q and P_{Node}, i.e., $D^{\text{LB}}_{\text{tns}}(q, P_{\text{Node}})$. Let P_{node} point to leaf(q) and P_{LB} be $D^{\text{LB}}_{\text{tns}}(q, P_{\text{Node}})$ (line 7), and then visit $\mathcal{ST}$ in a bottom-up manner (lines 8–23). If HG is empty, the adjust function is called to move P_{node} to its parent node and update P_{LB} accordingly (line 10). The adjust function will also process each unvisited child nodes of new P_{node}. The detail steps of the adjust function are shown in Algorithm 2.

Next, a tuple (c, dis) is popped-out from HG. Note that (c, dis) is the head element of HG, and HG is ordered by the (lower bound of) textual–numeric–spatial distances of its elements from query q. If dis, which is the (lower bound of) distance of head element c from query q, is larger than P_{LB}, then the query answer may be existed in the parent node of P_{node}, thus the adjust function is called to move P_{node} to its

Algorithm 1: A^2S^2KG Algorithm

```
input : STAG-tree $\mathcal{STAG}$, $A^2$SKIV query $q$=($q.W$, $q.V$, $q.L$, $k$)
output: Set $S_{result}$
1  begin
2      $S_{result}$=∅; float $D_{tsk} = +\infty$; $HG = \emptyset$;
3      Locate the leaf node (subgraph) leaf($q$) where $q$ lies;
4      for each object $o \in$ leaf(q) do
5          if $D_{tns}(q, o) \leq D_{tsk}$ then
6              $HG$.push($O, D_{tns}(q, o)$); //update $D_{tsk}$ accordingly;
7      $P_{Node}$=leaf($q$); $P_{LB} = D^{LB}_{tns}(q, P_{Node})$;
8      while $|S_{result}| < k$ && ($HG \neq \emptyset$ || $P_{Node} \neq R_0$) do
9          if $HG$=∅ then
10             Adjust($P_{Node}, P_{LB}, HG$);
11         ($c$, $dis$)=HG.pop();
12         if $dis > P_{LB}$&$P_{node} \neq R_0$ then
13             Adjust($P_{Node}, P_{LB}, HG$);
14         else
15             if $c$ is an object then
16                 insert c into $S_{result}$;
17             else
18                 if $c$ is a nonleaf subgraph then
19                     for each unvisited child node $s \in c$ do
20                         Gjudge($s$);
21                 else
22                     for each object $o \in c$ do
23                         Ojudge($o$);
```

Algorithm 2: Adjust Function

```
input : $P_N$, $P_{LB}$, $HG$
output: $P_{LB}$
1  begin
2      $P_{Node}$=$P_{Node}$.Parent;
3      for each unvisited child node $s$ of $P_{Node}$ do
4          $D_{tns}$=Gjudeg($s$);
5          if $D_{tns} < P_{LB}$ then
6              $P_{LB}$=$D_{tns}$;
```

parent node and update P_{LB} accordingly (line 13). Otherwise (dis $<= P_{\text{LB}}$), there are three cases: 1) c is an object, then c is a result object since c is the object with the minimum textual–numeric–spatial distance (lines 15 and 16); 2) c is a nonleaf subgraph, then for each unvisited subgraph s of c, function Gjudge (shown in Algorithm 3) is called to process each s; and 3) c is a leaf subgraph, then for each object o of c, function Ojudge is called to process o.

The detailed steps of the *Adjust* function are shown in Algorithm 2. It first moves P_{node} to its parent node (line 2). Then, for each unvisited child node s of P_{node}, the Gjudge function (Algorithm 3) is called to check if s possibly contains result objects, and if true, we calculate the lower bound of the textual–numeric–spatial distance between s and query q, i.e., $D^{\text{LB}}_{\text{tns}}(q, s)$. Finally, P_{LB}, which keeps the minimum value of $D^{\text{LB}}_{tns}(q, s)$ for all the child nodes of P_{node}, is returned.

The pseudocode of the *Gjudge* function is shown in Algorithm 3. For node s, *Gjudge* uses Lemmas 1–3, respectively, to check if s is a qualified subgraph, otherwise, s is safely pruned and 1, which is the uppermost limit of $D_{\text{tns}}(q, s)$, is returned. If s is not pruned, we: 1) calculate $D^{\text{LB}}_{\text{tns}}(q, s)$ and 2) push s together with $D^{\text{LB}}_{\text{tns}}(q, s)$ into HG (update D_{tsk} accordingly), and return $D^{\text{LB}}_{\text{tns}}(q, s)$ if $D^{\text{LB}}_{\text{tns}}(q, s) \leq D_{\text{tns}}(q, o_k)$; otherwise, s is pruned by Lemma 4 and 1 is returned, since if $D^{\text{LB}}_{\text{tns}}(q, s) > D_{\text{tns}}(q, o_k)$, s cannot contain any result object. The processing step of the Ojudge function is similar to that of the Gjudge function, and so we omit the discussion for space limitation.

Time Complexity Analysis: Finally, we discuss the time complexity of the A^2S^2KG algorithm. Given an object o and a top-k A^2SKIV query $q = (q \cdot W, q \cdot V, q \cdot L, k)$, o is a candidate result object for q if: 1) $o \cdot V$ contains all the numeric attributes of $q \cdot V$, i.e., $\forall q \cdot V \cdot A_j$, $1 \leq i \leq m$, $\exists o \cdot V \cdot A_i = q \cdot V \cdot A_j$, whose probability can be represented as $\text{Pr}_{\text{AM}}(o)$ and 2) $\exists q_j \in q \cdot W$, q_j.signature $\cap$ $\mathcal{G}_i$.signature $\neq \emptyset$, whose probability can be represented as $\text{Pr}_{\text{KM}}(o)$.

Thus, the total probability of an object o being a candidate object of q, $\text{Pr}_{\text{cand}}(o)$, equals $\text{Pr}_{\text{AM}}(o) * \text{Pr}_{\text{KM}}(o)$. Note here k result objects are required, the total number of objects visited is $(k/[\text{Pr}_{\text{cand}}(o)])$. For each object o being visited, the time costs for computing its $D_{\text{td}}(q, o)$ and $D_{\text{nd}}(q, o)$ are $O(|q \cdot W| * |o.\text{tags}|)$ and $O(|q \cdot V| * |o \cdot V|)$, respectively. Assume W_{dis} and V_{dis} are the total numbers of distinct keywords and distinct numeric attributes in the system, respectively. Thus, the time complexity for textual and attribute matching in query processing is $O((k/[\text{Pr}_{\text{cand}}(o)]) * (W^2_{\text{dis}} + V^2_{\text{dis}}))$.

To estimate the value of $\text{Pr}_{\text{AM}}(o)$, assume m and n are the numbers of numeric attributes for o and q, respectively, and $m \geq n$, we have, $\text{Pr}_{\text{AM}}(o) = C^n_n * ([C^{m-n}_{V_{\text{dis}}-n}]/[C^m_{V_{\text{dis}}}]) = ([C^{m-n}_{V_{\text{dis}}-n}]/[C^m_{V_{\text{dis}}}])$. Here, C^i_j is the number of combinations of taking i elements from a set of j.

For object o and query q, it is difficult to calculate the exact value of $\text{Pr}_{\text{KM}}(o)$, thus we use the probability of o.tags containing at least one keyword in $q \cdot W$ to approximate $\text{Pr}_{\text{KM}}(o)$. Assume m and n are the numbers of keywords for o and q, respectively, and $m \geq n$. Thus, we have

$$\text{Pr}_{\text{KM}}(o) \approx \begin{cases} 1 - \frac{C^m_{W_{\text{dis}}-n}}{C^m_{W_{\text{dis}}}}, & \text{if } m \leq W_{\text{dis}} - n \\ 1, & \text{otherwise.} \end{cases}$$

Since we use the G-tree to compute the shortest path distances, the time cost for computing $D_{\text{tr}}(q, o)$ is $O(\tau * \log \tau + \log_f(|V|/\tau) * \log^2_2 f * |V|)$ [37]. To sum up, the time complexity of the A^2S^2KG algorithm is $O([k/(\text{Pr}_{\text{cand}}(o))] * (W^2_{\text{dis}} + V^2_{\text{dis}}) + \tau * \log \tau + \log_f(|V|/\tau) * \log^2_2 f * |V|)$.

VI. Support Attributes With Interval Values

In real applications, the attribute value of an object is not necessarily a specific value, but usually an interval of values. In this section, we discuss extending our STAG-tree index to handle this situation, where the attribute value of the object is an interval of values.

Algorithm 3: Gjudge Function

```
input : $\mathcal{STAG}$, A$^2$SKIV query $q$, node $s$
output: $D_{tns}^{LB}(q, s)$
begin
    if $d_N^{\min}(q, s) > -\frac{1}{\rho}\ln(\frac{2}{\frac{D_{tsk}}{\gamma}+1} - 1)$ then
        return 1;    //$s$ is pruned by Lemma 1;
    if $\forall q_k \in q.W,\ q_k.signature \cap s.signature = \emptyset$ then
        return 1;    //$s$ is pruned by Lemma 2;
    if $\exists A_k \in q.V$, such that $\text{Max}(A_k)(or\text{Min}(A_k))$ for $\mathcal{G}_i.SGA$ equals $+\infty$ then
        return 1;    //$c$ is pruned by Lemma 3;
    if $D_{tns}^{LB}(q, s) > D_{tsk}$ then
        return 1;    //$s$ is pruned by Lemma 4;
    else
        $HG$.push($c$, $D_{tns}^{LB}(q, s)$);  //update $D_{tsk}$ accordingly;
        return $D_{tns}^{LB}(q, s)$;
```

A. Modification of Numeric Distance Calculation

First, we modify the calculation of the numeric distance between query $q = (q \cdot W, q \cdot V, q \cdot L, k)$ and object $o = (o.\text{tags}, o \cdot V, o \cdot L)$, for objects with attributes of interval values (IV for short). Remember that numeric attribute distance refers to the degree of difference between the values of query q and object o under the same numeric attribute.

For query q and object o, the numeric distance d_k between q and o for attribute A_k of interval values can be expressed as follows:

$$d_k = \begin{cases} +\infty, & \text{if } o.A_k \text{ not exists} \\ 0, & \text{if } o.A_k.IV \subseteq q.A_k.IV \\ M_k, & \text{if } o.A_k.IV \text{ does not intersect with } q.A_k.IV \\ |o.A_k.IV| - |o.A_k.IV \cap q.A_k.IV|, & \text{otherwise.} \end{cases} \quad (6)$$

Then, by using (3), we normalize each noninfinite numeric attribute distance to range [0, 1], and comprehensively consider each noninfinite numeric attribute distance to calculate the total numeric distance between q and o.

Note: 1) $M_k = \text{Max}(A_k)\text{-Min}(A_k)$; 2) if there is any query attribute not existing in $o \cdot V$, the numeric distance between q and o, i.e., $D_{\text{nd}}(q, o)$, equals $+\infty$.

B. Index Modification

Next, we discuss extending STAG-tree index to support queries and objects with attributes of interval values. In particular, the A-ref part needs to be modified to accommodate the interval values of numeric attributes for objects and queries.

In Section IV-A, for each numeric attribute A_k $(1 \le k \le n)$, we use $[k-1, k)$ to represent attribute-value range of A_k. Moreover, we map $o_i.V_k$, which is the attribute value for A_k of object o_i within $\mathcal{G}_i$, to the value range of A_k by the key $\text{y}(o_i^k) = ([o_i.V_k]/M_k) + k - 1 (1 \le k \le n)$. To accommodate the interval values for attribute A_k, we use $o_i.V_k.L$ and $o_i.V_k.R$ to represent the left and right bound of the interval values for $o_i.A_k$, respectively. Then, map $o_i.V_k.L$ and $o_i.V_k.R$ to the value range of A_k by the key $\text{y}(o_i^k.L) = ([o_i.V_k.L]/M_k) + k - 1$ and $\text{y}(o_i^k.R) = ([o_i.V_k.R]/M_k) + k - 1 (1 \le k \le n)$, respectively.

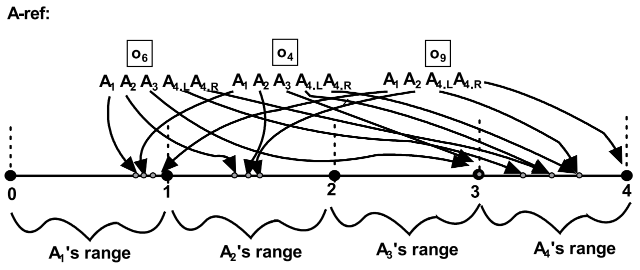

A-ref:
o_6
o_4
o_9
A_1 A_2 A_3 $A_{4.L}$ $A_{4.R}$
A_1 A_2 A_3 $A_{4.L}$ $A_{4.R}$
A_1 A_2 $A_{4.L}$ $A_{4.R}$
0
1
2
3
4
A_1's range
A_2's range
A_3's range
A_4's range

Fig. 6. A-ref modification.

TABLE II
CHARACTERISTICS OF THE DATA SETS

Attribute	FL	CAL
Number of vertices	1,070,376	21,048
Number of edges	1,356,399	21,693
Number of objects	10M	200k
Avg. number of keywords in objects	6.1	5.2
Avg. number of attributes in objects	4.3	4.1
Number of queries	1k	1k

Example 4: As shown in Fig. 6, we add the fourth numeric attribute, i.e., A_4 (business hours), for the objects in the system, and the value range of A_4 is [3, 4). Assume that the business hours for o_6 are from 8:00 to 12:00, and $M_4 = 24$ since there are 24 h in a day. Thus, we have $\text{y}(o_6^4.L) = (8/24) + 4 - 1 = 3.33$, and $\text{y}(o_6^4.R) = (12/24) + 4 - 1 = 3.50$.

The query processing steps are similar to that in Section V, except for the calculation of $D_{\text{nd}}(q, o)$.

VII. Performance Evaluation

A. Experimental Settings

1) Data Sets: We use two data sets, Florida (FL for short) and California (CAL for short), to test the performance of the proposed methods. FL and CAL consist of the traffic network, the users, and points of interest (POIs) of Florida and California, respectively. The object information for CAL comes from the Geographic Names Information System in the United States (geonames.usgs.gov/domestic). Each object includes an object ID, a textual description, and a location within the road traffic network. For the FL data set, we use the objects extracted from Twitter (www.twitter.com), and each object includes an object ID, a Twitter message, a time of publication, and a location in the Florida transportation network. The detail information of FL and CAL is shown in Table II.

2) Queries: To evaluate the performance of A^{2}SKIV, we generate a set queries, including locations, keywords, and attribute-value pairs. The keywords and attributes of A^{2}SKIV queries are also obtained from Twitter. In addition, attribute values are randomly selected and range from 1 to 1000. The number of query keywords and query attributes ranges from 1 to 5 and 1 to 4, respectively, with a default value of 2.

3) Algorithms: Our STAG-tree-based method (STAG for short) will be compared with two baseline methods, DBM [26]

TABLE III
EVALUATION PARAMETERS USED IN THE EXPERIMENT

Parameter	Values
Object cardinality ($\|D\|$)	FL: 2,4,**6**,8,10 (M) CAL: 10,50,**100**,150,200 (K)
Number of query results (k)	5,10,**15**,20,25
Number of query keywords ($\|q.W\|$)	1,**2**,3,4,5
Number of query attributes ($\|q.V\|$)	1,**2**,3,4
Preference parameter α	0.1,0.2,**0.33**,0.4,0.5
Preference parameter β	0.1,0.2,**0.33**,0.4,0.5
Preference parameter γ	0.1,0.2,**0.33**,0.4,0.5

and ILM [29], in terms of memory consumption and processing time. Specifically, DBM is based on the Dijkstra method. Starting from query q, DBM performs network expansion for candidate objects and calculates the textual–numeric–spatial distance of the object o encountered from query q, i.e., $D_{\text{tns}}(q, o)$. In order to accelerate the calculation of long road network distance, a multihop distance labeling scheme is adopted. ILM is an inverted-list-based scheme. For each keyword w, let the set of n-grams [32] contained in w be S_w. Thus, for object o, we have $S_o = \cup_{w\in o.\text{tags}} S_w$. For each n-gram ζ, a list l_ζ containing the ID of objects, whose n-grams contain ζ, can be obtained. For each query keyword $q_i \in q \cdot W$, S_{q_i} is computed, and then by using the heap algorithm [38], the object lists l_{ζ_j}s (for each $\zeta \in S_{q_i}$) are merged, to get a new list l_{q_i} of objects for q_i, whose objects are sorted in descending order of $|S_{q_i} \cap S_o|$. Thus, the objects sharing no common n-gram with q can be safely pruned. Similarly, for each query attribute A_i, we also have a list l_{A_i} containing the ID of objects which contain attribute A_i. Thus, the objects do not contain all the attributes $A_i \in q \cdot V$ are ignored.

B. Efficiency Measurement

This section evaluates the performance of three methods by varying the object cardinality, number of query results (k), number of query keywords, number of query attributes, and the values of preference parameters (α, β, and γ). The memory space for query processing is also studied. The main parameters and their values are shown in Table III.

1) Memory Consumption: The memory consumption of three methods is shown in Fig. 7, which increases as the number of objects increases. The more objects, the more storage space they take up. Generally speaking, STAG and ILM consume more memory resource than that of DBM for both FL and CAL data sets. For ILM, the reason is that it builds an inverted list for each keyword and attribute, and the object IDs store multiple copies in inverted lists. As for STAG, we build T-ref and A-ref part to keep the textual and numeric information of each object.

2) Effect of $|D|$: The running time of methods with respect to the number of objects in the system is shown in Fig. 8. It is observed that STAG outperforms its competitors. On average, the STAG-based approach is about 1.87× (17.1×, resp.) faster in processing time than the compared ILM (DBM, resp.) method. It is due to the fact that STAG can prune huge amounts of unpromising objects based on network distance, textual similarity, and attribute similarity, simultaneously. Fig. 8 also

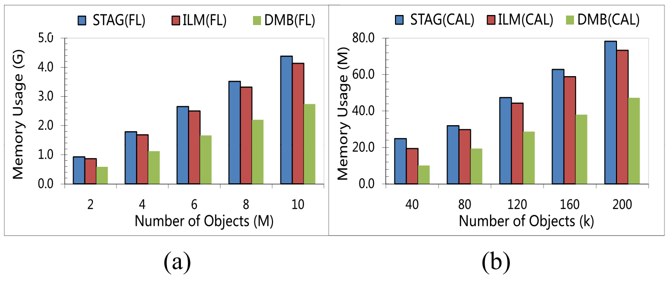


Fig. 7. Effect of $|D|$ on memory consumption. (a) FL data set. (b) CAL data set.

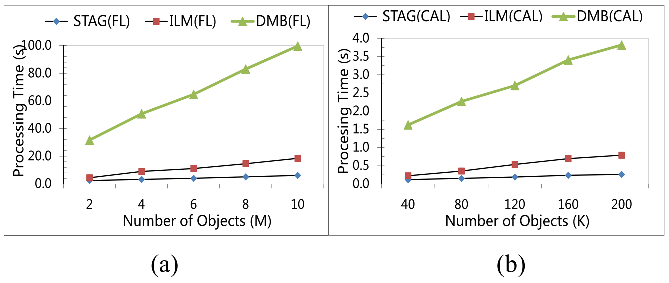


Fig. 8. Effect of $|D|$ on processing time. (a) FL data set. (b) CAL data set.

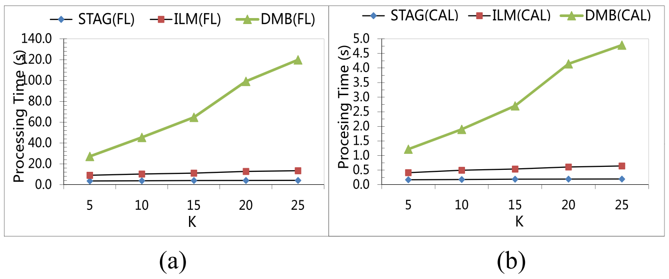


Fig. 9. Effect of k on processing time. (a) FL data set. (b) CAL data set.

shows that the running time of three methods increases as the object cardinality increases. It is natural since more related objects need to be considered when there are more objects in IoV. Moreover, STAG and ILM are much more scalable on the FL and CAL data sets than DBM, because DBM checks objects in the order being encountered. On the contrary, STAG and ILM arrange the objects according to keywords and attributes. Therefore, objects that do not contain all query attributes (or do not have any keyword similar to any query keyword) can be pruned securely, which makes both approaches more scalable than DBM.

3) Effect of k: The effect of value k (number of results wanted) on the running time of STAG, ILM, and DBM is evaluated. Fig. 9 shows that STAG significantly outperforms ILM and DBM, since it uses the STAG-tree index to prune large parts of unqualified objects. On the contrary, DBM performs the worst because it examines all the objects in the order being encountered and then computes their textual–numeric–spatial distance values. As for the stability of methods, when the value of k varies, all methods incur higher cost with larger k, because the larger k is, the more related objects need to be examined. For STAG, the increase of k value has no obvious effect on the performance due to the effective pruning scheme.

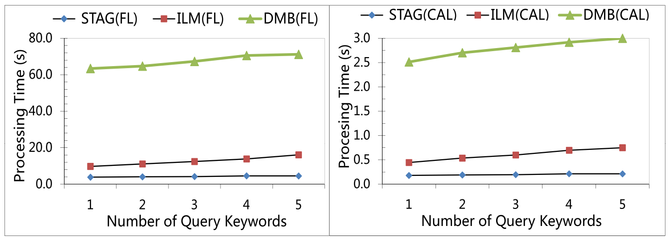


Fig. 10. Effect of $|q \cdot W|$ on processing time. (a) FL data set. (b) CAL data set.

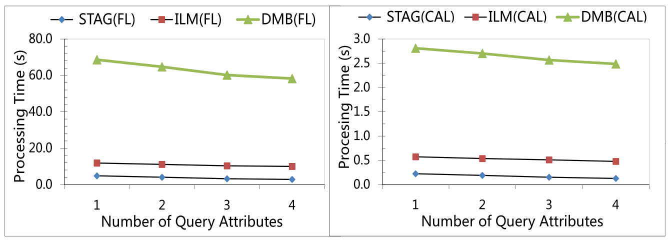


Fig. 11. Effect of $|q \cdot V|$ on processing time. (a) FL data set. (b) CAL data set.

4) Effect of $|q \cdot W|$: We also evaluate the query performance when the number of query keywords, $|q \cdot W|$, varies. Fig. 10 shows that the running time of all methods increases with the increment of $|q \cdot W|$. For STAG and ILM, the reason is that an object with any keyword similar to any query keyword has a chance to be one of the query results, thus more qualified objects need to be considered with larger $|q \cdot W|$. The processing time of DBM increases slightly with larger $|q \cdot W|$, because it requires more computation time to calculate the textual–numeric–spatial distance values of objects. Not surprisingly, STAG gets the best performance of three methods. For example, STAG requires only 36.4% (6.3%, resp.) processing time of ILM (DBM, resp.) when $|q \cdot W|$ equals 3 for FL data set.

5) Effect of $|q \cdot V|$: Now we continue to evaluate the impact of number of query attributes, $|q \cdot V|$, on performance of three schemes. Fig. 11 shows that all methods incur less processing time with larger query attributes. The reason is twofold. On the one hand, a candidate object is needed to contain all query attributes, and the more query attributes there are, the fewer eligible objects there are. On the other hand, the calculation of numeric distance values for candidate objects is little more difficult with more query attributes. Overall, the former outweighs the latter, so the total processing cost of the methods decreases as the number of query attributes increases. It is worth noting that the decreasing tendency of STAG is more obvious than its competitors due to its significant pruning ability, i.e., most of the cells (subgraphs) in STAG tree can be ignored as more attributes are queried.

6) Effect of α, β, and γ: Parameters α and β control the importance of textual and numeric similarity between queries and objects, respectively. When the value of α or β changes separately, there is no fixed impact pattern on the performance of query processing. As a result, we do not give the results of α and β for effective measurement, and only show the impact of γ on query efficiency. Note that varying the value of γ means varying the sum of α and β. Fig. 12 gives the query performance of these three methods for different γ values. Again, our STAG significantly outperforms its competitors. On average, it incurs only 38.0% (6.5%, resp.) query time of ILM (DBM, resp.) for the CAL data set. As for the stability of methods when the value of γ varies, three methods incur lower cost with larger γ, since larger γ means the spatial proximity between the query and objects becomes more important, thus the candidate objects may locate within a more concentrate range and fewer relevant objects need to be considered.

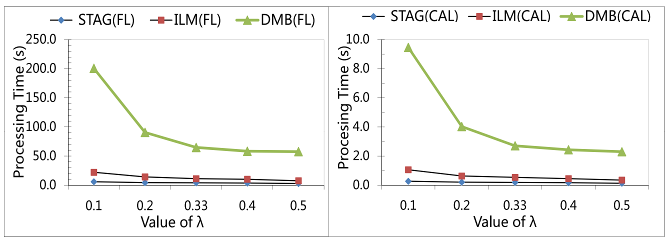


Fig. 12. Effect of γ on processing time. (a) FL data set. (b) CAL data set.

VIII. Conclusion

This article formulated and solved fog-computing-based A^{2}SKIV in IoV. A fog-based network structure FCV is adopted to improve query processing efficiency and reduce query feedback time. To deal with A^{2}SKIV queries, a two-level hybrid index STAG-tree is proposed, whose first level is a G-tree which accelerates the calculation of the network distance between objects and the query, and whose second level is the textual and numeric component which efficiently organizes the information of objects within the subgraphs of traffic network in IoV. In addition, several lemmas are presented to prune a huge number of unqualified textual–spatial objects, and an efficient top-k A^{2}SKIV query processing algorithm was presented. The effectiveness of the proposed index and query processing algorithm is verified by extensive experimental evaluation using real and composite data sets. The results also showed that the proposed scheme is effective in applications, such as mobile search and targeted location-aware advertising in IoV.

References

[1] R. Yu, Y. Zhang, S. Gjessing, W. Xia, and K. Yang, "Toward cloud-based vehicular networks with efficient resource management," *IEEE Netw.*, vol. 27, no. 5, pp. 48–55, Sep./Oct. 2013.
[2] J. Li, J. Jin, Y. Dong, and H. Zhang, "Virtual fog: A virtualization enabled fog computing framework for Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 121–131, Feb. 2018.
[3] M. Aazam and E.-N. Huh, "Fog computing and smart gateway based communication for cloud of things," in *Proc. Int. Conf. Future Internet Things Cloud*, 2014, pp. 464–470.
[4] K. Dan, K. Piratla, and C. J. Matthews, "Towards sustainable water supply: Schematic development of big data collection using Internet of Things," *Procedia Eng.*, vol. 118, pp. 489–497, Sep. 2015.
[5] G. Li, C. Zhou, J. Li, and B. Guo, "Maintaining data freshness in distributed cyber-physical systems," *IEEE Trans. Comput.*, vol. 68, no. 7, pp. 1077–1090, Jul. 2019.

[6] C. Yu, B. Lin, P. Guo, W. Zhang, S. Li, and R. He, "Deployment and dimensioning of fog computing-based Internet of Vehicle infrastructure for autonomous driving," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 149–160, Feb. 2019.

[7] Y. Zhang, C.-Y. Wang, and H.-Y. Wei, "Parking reservation auction for parked vehicle assistance in vehicular fog computing," *IEEE Trans. Veh. Technol.*, vol. 64, no. 4, pp. 3126–3139, Apr. 2019.

[8] Z. Lamb and D. Agrawal, "Analysis of mobile edge computing for vehicular networks," *Sensors*, vol. 19, no. 6, pp. 1303–1323, 2019.

[9] D. Zhang, K.-L. Tan, and A. K. H. Tung, "Scalable top-*k* spatial keyword search," in *Proc. Int. Conf. Extend. Database Technol.*, 2013, pp. 359–370.

[10] Q. Zhou, G. Li, J. Li, and C. Deng, "Execution-efficient response time analysis on global multiprocessor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 12, pp. 2785–2797, Dec. 2018.

[11] Y. E. Carlsson, "Keyword search on spatial network databases: Road network indexing for efficient query processing," M.S. thesis, Dept. Comput. Sci., Norwegian Univ. Sci. Technol., Trondheim, Norway, pp. 1–63, 2011.

[12] Q. Zhou, G. Li, J. Li, L. C. Shu, C. Zhang, and F. Yang, "Dynamic priority scheduling of periodic queries in on-demand data dissemination systems," *Inf. Syst.*, vol. 67, pp. 58–70, Jul. 2017.

[13] L. Kuang, L. T. Yang, J. Chen, H. Fei, and C. Luo, "A holistic approach to distributed dimensionality reduction of big data," *IEEE Trans. Cloud Comput.*, vol. 6, no. 2, pp. 506–518, Apr.–Jun. 2018.

[14] 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, 2015.

[15] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6774–6785, Aug. 2019.

[16] F. Kong, J. Li, B. Jiang, and H. Song, "Short-term traffic flow prediction in smart multimedia system for Internet of Vehicles based on deep belief network," *Future Gener. Comput. Syst.*, vol. 93, pp. 460–472, Apr. 2019.

[17] G. Mastorakis, E. Pallis, C. X. Mavromoustakis, L. Shu, and J. J. P. C. Rodrigues, "Special issue on multimedia services provision over future mobile computing systems," *IEEE Syst. J.*, vol. 12, no. 1, pp. 12–15, Mar. 2018.

[18] M. Mukherjee, L. Shu, and D. Wang, "Survey of fog computing: Fundamental, network applications, and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 1826–1857, 3rd Quart., 2018.

[19] C. Sheng, W. Liao, C. Luo, L. Ming, and L. Pan, "CRIL: An efficient online adaptive indoor localization system," *IEEE Trans. Veh. Technol.*, vol. 66, no. 5, pp. 4148–4160, May 2017.

[20] S. Salinas, C. Luo, X. Chen, W. Liao, and P. Li, "Efficient secure outsourcing of large-scale sparse linear systems of equations," *IEEE Trans. Big Data*, vol. 4, no. 1, pp. 26–39, Mar. 2018.

[21] M. Aazam and E.-N. Huh, "Dynamic resource provisioning through fog micro datacenter," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops*, 2015, pp. 105–110.

[22] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016.

[23] Y. Yuan, X. Lian, L. Chen, J. Yu, G. Wang, and Y. Sun, "Keyword search over distributed graphs with compressed signature," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 6, pp. 1212–1225, Jun. 2017.

[24] X. Miao, Y. Gao, G. Su, and C. Gang, "On efficiently answering why-not range-based skyline queries in road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 9, pp. 1697–1711, Sep. 2018.

[25] 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. Int. Conf. Data Eng.*, 2019, pp. 986–997.

[26] L. Chang, J. X. Yu, L. Qin, H. Cheng, and M. Qiao, "The exact distance to destination in undirected world," *VLDB J.*, vol. 21, no. 6, pp. 869–888, 2012.

[27] Y. Gao, X. Qin, B. Zheng, and G. Chen, "Efficient reverse top-*k* Boolean spatial keyword queries on road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 5, pp. 1205–1218, May 2015.

[28] J. Zhao, Y. Gao, C. Gang, and C. Rui, "Towards efficient framework for time-aware spatial keyword queries on road networks," *ACM Trans. Inf. Syst.*, vol. 36, no. 3, pp. 1–48, 2017.

[29] D. Zhang, Y. Li, X. Cao, J. Shao, and H. T. Shen, "Augmented keyword search on spatial entity databases," *VLDB J.*, vol. 27, no. 2, pp. 225–244, 2018.

[30] Y. Li *et al.*, "Intelligent augmented keyword search on spatial entities in real-life Internet of Vehicles," *Future Gener. Comput. Syst.*, vol. 94, pp. 697–711, May 2019.

[31] T. Abeywickrama, M. A. Cheema, and A. Khan, "K-SPIN: Efficiently processing spatial keyword queries on road networks," *IEEE Trans. Knowl. Data Eng.*, to be published.

[32] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, "Approximate string joins in a database (almost) for free," in *Proc. VLDB*, 2001, pp. 491–500.

[33] S. Alsubaiee, A. Behm, and L. Chen, "Supporting location-based approximate-keyword queries," in *Proc. SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, 2010, pp. 61–70.

[34] X. Wang, X. Ding, A. K. H. Tung, and Z. Zhang, "Efficient and effective KNN sequence search with approximate n-grams," *Proc. VLDB Endow.*, vol. 7, no. 1, pp. 1–12, 2013.

[35] B. Zheng, N. J. Yuan, K. Zheng, X. Xie, S. Sadiq, and X. Zhou, "Approximate keyword search in semantic trajectory database," in *Proc. Int. Conf. Data Eng.*, Seoul, South Korea, 2015, pp. 975–986.

[36] B. Yao, F. Li, M. Hadjieleftheriou, and K. Hou, "Approximate string search in spatial databases," in *Proc. Int. Conf. Data Eng.*, 2010, pp. 545–556.

[37] R. Zhong, G. Li, K.-L. Tan, and L. Zhou, "G-Tree: An efficient index for KNN search on road networks," in *Proc. ACM Int. Conf. Inf. Knowl. Manag.*, 2013. pp. 39–48.

[38] S. Sarawagi and A. Kirpal, "Efficient set joins on similarity predicates," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2004, pp. 743–754.

Yanhong Li received the Ph.D. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2011.

She is currently a Professor with the College of Computer Science, South Central University for Nationalities, Wuhan. She has published over 25 papers in international journals and conferences in the areas of mobile computing and wireless networks. Her research interests include spatial information and communication and multimedia network technology.

Rongbo Zhu (Member, IEEE) received the Ph.D. degree in communication and information systems from Shanghai Jiao Tong University, Shanghai, China, in 2006.

He is currently a Professor with the College of Computer Science, South Central University for Nationalities, Wuhan, China. He has published over 60 papers in international journals and conferences in the areas of wireless networks and mobile computing.

Prof. Zhu is an Associate Editor of IEEE ACCESS and the *International Journal of Radio Frequency Identification Technology and Applications*.

Shiwen Mao (Fellow, IEEE) received the Ph.D. degree in electrical and computer engineering from Polytechnic University, Brooklyn, NY, USA.

He was the McWane Associate Professor with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL, USA, from 2012 to 2015, where he is currently the Samuel Ginn Distinguished Professor and the Director of the Wireless Engineering Research and Education Center. His research interests include wireless networks, multimedia communications, and smart grid.

Ashiq Anjum (Member, IEEE) received the Ph.D. degree in computer science from European Organization for Nuclear Research, Geneva, Switzerland, in 2007.

He is currently a Professor of distributed systems with the University of Derby, Derby, U.K. He currently investigates high-performance distributed platforms to efficiently process video and genomics data. He has more than 70 international academic publications. His research interests include data intensive distributed systems, block chain, Internet of Things, and high performance analytics platforms.

Prof. Anjum is a member of the British Computer Society, ACE, and SIGHPC. He is the Fellow of the Higher Education Academy and the Champion of European Grid Infrastructure.