IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 26, 2016, accepted August 10, 2016, date of publication August 25, 2016, date of current version September 16, 2016.

Digital Object Identifier 10.1109/ACCESS.2016.2602702

Skyline Path Queries With Aggregate Attributes

YI-CHUNG CHEN[1], (Member, IEEE), AND CHIANG LEE[2]
[1]Department of Information Engineering and Computer Science, Feng Chia University, Taichung 407, Taiwan
[2]Institute of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan

Corresponding author: Y.-C. Chen (chenyic@fcu.edu.tw)

This work was supported by the Ministry of Science and Technology, Taiwan, under Contract MOST 104-2119-M-035-002, Contract MOST 105-2634-E-035-001, and Contract MOST 104-2221-E-006-251.

ABSTRACT The skyline path query is a novel extension of skyline queries. A skyline path query retrieves a set of non-dominated paths from origin s to destination t. On a road network using multiple path criteria, such as the distance, travel time, and number of travelers on a path, this paper extends the concept of skyline path query by considering a new type of criteria referred to as the aggregate attribute of paths. The method used for calculating this type of criteria is very different from that of existing criteria, and this can have a notable effect on the processing of ordinary skyline path queries. This paper defines the aggregate attributes of paths, discusses the impact of aggregate attributes on skyline path queries, and proposes a novel index tree with an intelligent algorithm to find the skyline path while taking aggregate attributes into account. Experiments demonstrate the effectiveness and efficiency of the proposed algorithm.

INDEX TERMS Databases, query processing, skyline query, path planning, indexed tree.

I. INTRODUCTION

Path planning is among the most common location-based services, used for commercial purposes such transport management and resource allocation, as well as in everyday life, such as planning routes using online maps (e.g. Bing Maps [24], Google Maps [25], MapQuest [26], and Yahoo! Maps [27]). Online maps differ in their path search criteria (examples include the shortest distance, the shortest travel time, or the fewest transfers on public transport), and searches are usually limited to a single criterion. This limitation often makes it difficult to obtain a suitable solution. The shortest path may require that users cut through areas prone to congestion, whereas the path with the shortest travel time may require that users take toll highways. In response, Tian *et al.* [21] and Kriegel *et al.* [15] proposed the concept of the skyline path query. Given an undirected graph with origin s and destination t, we use path p to connect s and t, where p possesses d attribute values $[w_1, w_2, \ldots, w_d]$. If p is not worse than another path p' in all attribute values and is better in at least one attribute value, then p is said to dominate p'. A skyline path query returns all paths that are not dominated by other paths.

Figure 1 is an example of skyline path, where each vertex is a bus station. Edges between vertices indicate that a bus route runs between the bus stations, wherein the two attribute values associated with each edge represent the travel time in minutes and expense in dollars between the bus stations.

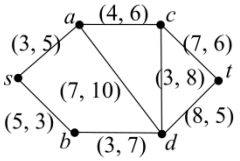

FIGURE 1. A skyline path example.

For example, edge (s, a) has values (3, 5), which indicates that taking a bus from s to a requires 3 minutes and 5 dollars. The paths in Figure 1 consists of multiple edges, wherein the attribute values for the paths are obtained by summing the attribute values of each constituent edge. For example, path p_1 is $< s, a, c, t >$, which indicates that this path involves vertices s, a, c, and t. Figure 1 shows that taking a bus from s to t via p_1 would entail 3+4+7=14 minutes and 5+6+6=17 dollars. A user could employ a skyline path query to identify the path (bus route) from vertex s to t of the shortest time for the least expense. First, the skyline path query identifies all possible paths between s and t and then calculates the total time and expense required for each of these paths. The results of this query are tabulated in Table 1. This table shows that p_1 dominates p_2 because the time and expense of p_1 are both smaller than those of p_2. Moreover, p_1 cannot be compared with p_7, because the time of p_1 is smaller than that of p_7, despite the fact that the expense

TABLE 1. The paths from vertex *s* to *t* and their time cost and money cost.

Path	Time Cost	Money Cost	Path	Time Cost	Money Cost
p_1<s,a,c,t>	14	17	p_5<s,b,d,a,c,t>	26	32
p_2<s,a,c,d,t>	18	24	p_6<s,b,d,c,t>	18	24
p_3<s,a,d,c,t>	20	29	p_7<s,b,d,t>	16	15
p_4<s,a,d,t>	18	20			

of p_7 is smaller than that of p_1. Based on this conception of domination, the skyline path query in Table 1 returns p_1 and p_7, because only these two paths are not dominated by other paths.

TABLE 2. The number of customers who would travel between two vertices.

	s	a	b	c	d	t
s	-					
a	80	-				
b	20	10	-			
c	20	50	45	-		
d	10	15	20	20	-	
t	100	30	55	35	20	-

The above example illustrates the most common method used for the calculation of attribute values for a given path. However, the attribute values can also be obtained using other calculation methods. For example, the manager of the bus company may decide to establish new bus paths using Figure 1, where each vertex represents a bus station and the edges between vertices represents new bus paths that could be established between bus stations. The attribute values of each edge represent the time and expense involved in traveling between bus stations, respectively. The manager of the bus company might also consider the average number of passengers taking a given bus in either direction every day, as shown in the lower triangular matrix of Table 2. For example, edge (s, a) presents a value of 80, which denotes that an average of 80 passengers embark at station s and disembark at station a, or vice versa, every day. A bus traversing path $< s, a, c >$ in both directions could serve 80 passengers wishing to travel between stations s and a. Moreover, a bus traveling path $< s, a, c >$ could serve passengers traveling between stations s and a, s and c, and a and c. Thus, the total number of passengers taking this bus per day would be the sum of (s, a), (s, c), and (a, c), which is 150, as tabulated in Table 2. This calculation method makes it possible to determine the number of passengers traveling along all possible paths between station s and station t, as shown in Figure 1. These results are tabulated in Table 3. Based on the concept of domination, we can deduce from Table 3 that p_1, p_2, p_5, and p_7 are skyline paths. Among these paths, p_5 requires more time and expense than do the other paths, while carrying the greatest number of passengers, which makes it a skyline path.

Note that many applications, such as transportation and city planning, have the same problem that the manager of the bus company meets. For example, we could consider Table 2 as a representation of the average number of goods that are moved by an express company between different cities every day, and the manager of this company wants to arrange a fixed daily route that costs less in terms of time and money but moves more goods. Or Table 2 might present the number of cars moving between different cities and the government wants to design a highway to save money and time while serving more users. Both of these issues are effectively addressed by the proposed method.

TABLE 3. The paths from vertex *s* to vertex *t* and their time cost, money cost, and number of customers.

Path	Time Cost	Money Cost	Number of Customers
p_1<s,a,c,t>	14	17	315
p_2<s,a,c,d,t>	18	24	380
p_3<s,a,d,c,t>	20	29	380
p_4<s,a,d,t>	18	20	255
p_5<s,b,d,a,c,t>	26	32	580
p_6<s,b,d,c,t>	18	24	345
P_7<s,b,d,t>	16	15	225

The above examples show that the path attributes in an undirected graph can be calculated using two methods. The first method involves accumulating all edge values along a path, such as the time required to travel the path. This work denotes an attribute obtained using this method as an "edge attribute." The primary characteristic of edge attributes is that there is no change in the magnitude relationship between edge attribute values and different paths when arbitrary edges are added to these paths. For example, Figure 1 shows that the total time required is 7 minutes for path $< s, a, c >$ and 11 minutes for path $< s, b, d, c >$. When edge (c, t), which requires 7 minutes, is separately added to paths $< s, a, c >$ and $< s, b, d, c >$, their total required time increases to 14 and 18 minutes, respectively. The time for $< s, a, c, t >$ is 4 minutes less than the time required for $< s, b, d, c, t >$, as was true for their subpaths to which edge (c, t) was added.

The second method involves accumulating all vertex combination values along a path using a triangular matrix, and then calculating the sum of these values, such as the total number of passengers traversing a specific path. This work denotes an attribute obtained using this method as an "aggregate attribute." Unlike an edge attribute, the magnitude relationship of aggregate attribute values may change, even when the same edge is added to a different path. For example, in Figure 1, a bus traveling along path $< s, a, c >$ attracts passengers traveling between stations s and a, s and c, and a and c. Therefore, according to the triangular matrix of Table 2, the total number of passengers taking this path can be expressed as the sum of the passenger counts for the constituent subpaths: $(s, a) + (s, c) + (a, c) = 150$. Alternatively, the total number of passengers for path $< s, b, d, c >$ can be expressed analogously: $(s, b) + (s, d) + (s, c) + (b, d) + (b, c) + (c, d) = 135$, which is lower than the number of passengers for path $< s, a, c >$. However, when edge (c, t) is added to

these two paths, the passenger count for path $< s, a, c >$ must be augmented by the passenger counts between stations s and$t$, a and t, and c and t: $(s, a) + (s, c) + (a, c) + (s, t) + (a, t) + (c,t) = 315$. The passenger count for path $< s, b, d, c >$ must be augmented by the passenger counts between stations s and t, b and t, d and t, and c and t: $(s, b) + (s, d) + (s, c) + (b, d) + (b, c) + (c, d) + (s, t) + (b, t) + (d, t) + (c, t) = 345$. The arithmetic relationship between the passenger counts for paths $< s, a, c >$ and $< s, b, d, c >$ differ from that for the paths obtained by adding edge (c, t). The incorporation of such an attribute complicates skyline path queries and leads to a considerable increase in the number of search queries.

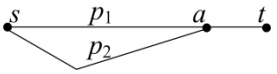

FIGURE 2. The example of the greedy algorithm.

Current skyline path algorithms [15], [21] can use only the edge attributes of road networks to identify skyline paths, which means that they are unable to identify skyline paths when road networks simultaneously possess both edge as well as aggregate attributes. This is primarily because conventional skyline path algorithms typically employ a greedy algorithm, which selects optimal solutions using a step-by-step approach from the origin to the destination. This approach is suitable only for the identification of optimal solutions from edge attribute values. In the following, we provide an example to clarify this situation using Figure 2, which denotes that p_1 and p_2 are feasible paths of vertices s and a. Suppose that the greedy algorithm determined that p_1 requires less time than does p_2. Then, any path with a minimum travel time starting from s, passing a, and arriving at t must include p_1 and not p_2, due to the fact that the magnitude relationship of edge attribute values along different paths do not change with the addition of new edges. Thus, when edge (a, t) is separately added to p_1 and p_2, the time required to travel from stations s to t via p_1 is still less that the path going via p_2. This example shows that optimal solutions calculated from edge attributes using the greedy algorithm remain optimal even after the addition of edges. However, the greedy algorithm is not suitable for aggregate attributes. Referring again to Figure 2, if p_1 were identified by the greedy algorithm as the path with the lowest aggregate attribute value between stations s and a, this may or may not hold after adding edge (a, t). This is because the magnitude relationship of aggregate attribute values on different paths may change with the addition of edges. This means that a greedy algorithm is unable to reliably identify the path with the lowest aggregate attribute value. To identify skyline paths in road networks that simultaneously possess edge attributes as well as aggregate attributes, the most naïve method would be to find all possible paths between an origin and a destination, and then separately calculate the edge attribute and aggregate attribute values of all paths. All of the paths would then have to undergo domination checks before the skyline paths could be obtained. This procedure is extremely time-consuming.

In this study, we sought to develop a novel algorithm to overcome this problem by dividing road networks into multiple regions. This enables the algorithm to determine skyline paths without having to identify all possible paths between an origin and a destination. Figure 3(a) illustrates a road network divided into regions. If the origin point is s, destination t can be reached by a path going through one of the following series of regions: $< r_s, r_1, r_2, r_t >$, $< r_s, r_1, r_2, r_3, r_4, r_t >$, $< r_s, r_3, r_2, r_t >$, or $< r_s, r_3, r_4, r_t >$. If the algorithm determines that no paths passing through r_3 and r_4 yield skyline paths, then the algorithm eliminates those paths, leaving only $< r_s,r_1,r_2,r_t >$ and $< r_s,r_3,r_2,r_t >$ for further analysis.

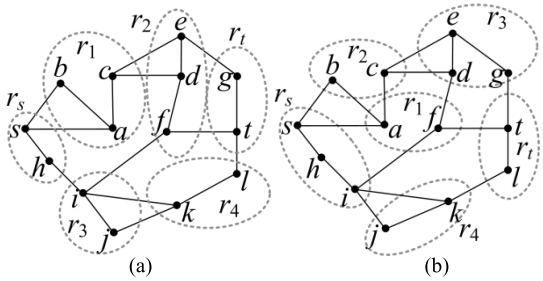

FIGURE 3. Example of a network being partitioned into regions in a way that (a) suits a shortest-path algorithm, (b) does not suit a shortest-path algorithm.

Care must be taken in the assignment of vertices to a region. Figure 3(b) presents an unsuitable assignment of vertices to regions, as illustrated by the path $< r_s, r_1, r_t >$, which supposedly involves a path by which to reach destination t from origin s, i.e, a potential skyline path. In actuality, $< r_s, r_1, r_t >$ cannot possess skyline paths because no edge connects vertices a and f in r_1, which suggests that no feasible paths exist between s and t, such that searching for skyline paths in $< r_s, r_1, r_t >$ is a waste of time. In contrast, any two vertices within a given region in Figure 3(a) are connected by at least one path lying entirely within that region. This means that a feasible path will inevitably be found among the various combinations of regions in Figure 3(a). Thus, the division-based method presented in Figure 3(a) is superior to that shown in Figure 3(b).

During the division of road networks, traditional R-tree [3], [9] and M-tree [7] solutions are plagued by the problem in which vertices in the same region may not be connected. These trees are able to place neighboring vertices in the same region; however, there is no mechanism by which to determine whether these vertices are connected in the road network, thereby reducing the search efficiency of the algorithm. Thus, we developed a novel tree (the one-hop tree), for the division of road networks into multiple regions and the indexing of vertices and regions within the road network. Using the one-hop tree to divide regions ensures that the vertices in each region are connected to other vertices in the same region, thereby saving time in the search for skyline paths. Finally, we conducted simulations to verify the efficiency and performance of the algorithm.

The remainder of this paper is organized as follows. Section II discusses related work on the identification of skyline paths. Section III defines the parameters used in this paper. Section IV presents the structure and the algorithms of the one-hop tree and outlined the proposed SPA algorithm. Section V presents simulation results. Section VI concludes the paper.

II. RELATED WORK

In this section, we introduce a number of previous works related to this paper, including those dealing with skyline queries and skyline queries on road networks.

A. SKYLINE QUERIES

Common skyline query algorithms include the Block Nested Loops Algorithm, Divide and Conquer Algorithm, Sort and Limit Skyline algorithm, and the Branch and Bound Skyline algorithm. In the following, we present a brief outline of the concepts underlying these four algorithms.

1) BLOCK NESTED LOOPS ALGORITHM (BNL)

The BNL algorithm [4] is the simplest and most straightforward skyline query method, the objective of which is to build a candidate skyline set. This algorithm compares each data point with every other point in the dataset. If it is not dominated, then it forms part of the skyline; if it is dominated, it is discarded. This comparison process is repeated for each point in the dataset. The resulting candidate skyline set constitutes the outcome of the skyline query. The BNL algorithm requires that any two data points in the database be scanned and tested for dominance; therefore, the time required for computation increases as the volume of data grows.

2) DIVIDE AND CONQUER ALGORITHM (DAC)

The DAC algorithm [4] breaks down data into groups and then conducts a skyline query in each group. The group results are combined into a final skyline query to obtain the ultimate outcomes. A number of data points can be eliminated during the initial grouping process; however, this method remain somewhat inefficient in recognizing and discarding irrelevant data points. This has prompted researchers to develop a number of other algorithms to identify and reject dominated points, in order to increase the processing speed of skyline queries.

3) SORT AND LIMIT SKYLINE ALGORITHM (SaLSa)

The SaLSa algorithm [2] uses a feature obtained from the raw data as a threshold value with which to filter and discard data points. The feature may be the total, minimum, or product of all data dimensions. SaLSa arranges the data sequentially based on the given feature. Before testing for dominance, the algorithm checks each point to determine whether it falls within that feature, which is essentially used as a skyline threshold. If it is below the threshold value, it must be tested for dominance. If it exceeds the threshold value, then subsequent points need not be tested for dominance. This approach makes it possible to discard all points above the threshold value quickly and easily.

4) BRANCH AND BOUND SKYLINE ALGORITHM (BBS)

The BBS algorithm [18] is currently the most popular skyline query algorithm. The BNL and DAC algorithms require that most of the data points be processed in order to complete the skyline query comparison. In contrast, BBS employs an index structure for the identification of skyline points, which reduces the number of points that must be tested in order to process a query. Figure 4(a) illustrates the first step in building the index, where M_0 is the root node comprising all of the data points. In Figure 4(b), the root node is divided into M_1 and M_2, wherein only M_1 includes skyline points. Thus, there is no need to process M_2. The deconstruction of M_1, as well as M_3 and M_4, is shown in Figure 4(c). This approach eliminates the need to process any points in M_2, which significantly reduces time and cost.

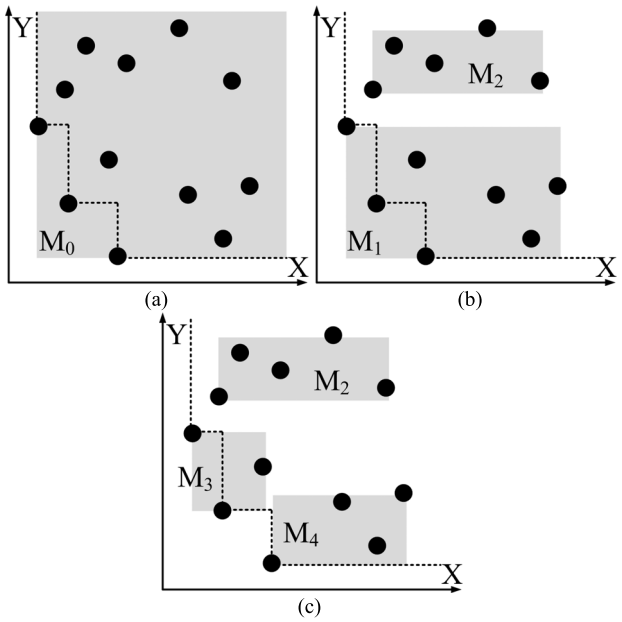

FIGURE 4. Example of the BBS algorithm (a) Initialization, (b) After M_0 is expanded, (c) After M_1 is expanded.

B. SKYLINE QUERIES ON ROAD NETWORKS

Previous studies in which skyline queries were used to search road networks can be classified into those endeavoring to identify skyline landmarks and those endeavoring to identify skyline paths.

1) IDENTIFYING SKYLINE LANDMARKS IN ROAD NETWORKS

Deng *et al*. [8] introduced the concept of searching for skyline landmarks in road networks. The skyline landmark query identifies landmarks that match user criteria when user is traveling on a road network. For example, when a user travels on a road network, skyline landmark query facilitates him/her in identifying affordable high-end restaurants that are nearby.

The algorithm proposed in this work characterizes landmark attributes as static or dynamic. Static attributes have fixed values, such as restaurant prices. Dynamic attributes have variable values, such as the distance between the user and a given landmark. The proposed algorithm first identifies static skyline landmarks based on their static attribute values. Then, when users perform searches, the algorithm identifies all dynamic skyline landmarks based on their dynamic attribute values, and combines search results with the static skyline landmarks. Finally, the algorithm can retrieve skyline landmarks that fit all attributes.

Huang and Jensen [13] proposed a different skyline landmark search concept from that of [8]. They argued that users' travel in road networks should be based on a previously established path. Thus, when users search for skyline landmarks, these landmarks should be determined based on a preset path, and not from the location of the user. The algorithm proposed in this work was similar to that proposed by Deng *et al*. [8], which used the concepts of static and dynamic attributes to identify skyline landmarks. The only difference between the algorithms is the attribute calculation method. Deng *et al*. [8] considered the distance between the landmark and the search location of the user; whereas the researchers of this work considers the distance between the landmark and the path preset by the user.

A new type of query referred to a continuous skyline query has recently been developed for the identification of skyline landmarks in road networks. This type of query precomputes range R for each landmark l to ensure that if a user u moves within R, then l will always be the skyline landmark for u [14]. Since the development of this method, numerous researchers have sought to modify the definitions in [14] to fit other applications that are more difficult to solve. For example, Son *et al.* [20] treated the distance on road network as the Manhattan distance in the development of Manhattan spatial skyline queries. Huang and He [12] considered uncertain dimensional values within a Euclidean space with moving objects, referring to it as the continuous possible K-nearest skyline query. Pan *et al.* [17] integrated the concept of probabilistic skyline queries with continuous skyline queries on road networks, and proposed a number of pruning methods. They reported that these innovations make this process more efficiently than other related algorithms.

2) IDENTIFYING SKYLINE PATHS IN ROAD NETWORKS

Tian *et al*. [21] introduced the concept of skyline paths. Their proposed algorithm would use the edge attributes of a road network to find all skyline paths between the user-specified starting vertex and destination. The algorithm would first determine a single skyline path between a starting vertex and destination whose summation of all attributes values is the lowest among all paths. Then, the algorithm would identify other skyline paths by (1) a greedy algorithm to find a possible relay vertex a between starting vertex s and destination t, and (2) calculating the actual attribute values of path $< s, a >$ and estimating the minimum attribute values of path $< a, t >$. If skyline path domination was present after adding the two values, then a cannot be part of a skyline path. In this instance, the algorithm again employs the greedy algorithm to identify other possible relay vertices or identify the next relay vertex following a.

Kriegel *et al*. [15] also employed the greedy algorithm to identify a possible relay vertex a between s and t. However, rather than employing the most naïve estimation method; Kriegel *et al*. [15] used a reference vertex x to support estimations. When users perform a search and provide s and t, the algorithm calculates the actual attribute values of path $< x, t >$. Then, whenever the greedy algorithm finds a relay vertex a, the proposed algorithm would estimate the various minimum attribute values of path $< a, x >$. In this instance, because a, x, and t can form a triangle, the algorithm can use $< x, t >$, $< a, x >$, and the triangle inequality to estimate the minimum attribute values of path $< a, t >$. By using such a method, they declared the method proposed in this work was faster than that proposed in the work of [21].

Many researchers have sought to extend the works in [15] and [21]. Aljubayrin *et al*. [1] discussed the problem of skyline trips on multiple POI categories. Hsu *et al*. [10] applied the idea of a skyline path to the planning of trips to overcome the conventional problem of obtaining multicriteria answers. Wen *et al*. [22] extended the works of [10] with the inclusion of personal keywords, and reported that they can provide results of greater accuracy. Yang *et al*. [23] combined GPS history data in their queries to help users plan their skyline route under time-varying uncertainty. Unfortunately, these works do not consider aggregate attributes in road networks, which makes them inapplicable to the problems addressed in this study.

III. DEFINITIONS

In this section, we define some of the parameters used in this work and specify our research goals.

Definition 1 (Vertices): $V = \{v_1, v_2, \ldots, v_{|\mathbf{V}|}\}$ denoting a collection of vertices in an undirected graph, where $|\mathbf{V}|$ represents the number of vertices. ■

Definition 2 (Edges): $\mathrm{E} = \{\mathrm{e1}, \mathrm{e2}, \ldots, \mathrm{e}|\mathrm{E}|\}$ denoting a collection of edges in an undirected graph, where |E| represents the number of edges. For all ei ∈ E, ei = (vj, vk), where vj, vk∈V are two vertices and vj ≠ vk. ■

Definition 3 (Edge Attributes of an Edge): For all $e_i \in E$, $Eattr(e_i) = \{w_{i1}, w_{i2}, \ldots, w_{im}\}$ denoting a collection of edge attributes of e_i, where m is the number of edge attributes. ■

Definition 4 (Aggregate Attributes of Two Vertices): For all $v_i, v_j \in V$, $Aattr(v_i, v_j) = \left\{u_{ij_1}, u_{ij_2}, ..., u_{ij_n}\right\}$ denoting a collection of aggregate attributes of v_i and v_j, where n is the number of aggregate attributes. Note that if $v_i = v_j$, then $u_{ij_1} - u_{ij_n}$ are all equal to 0. ■

Definition 5 (Path): A path $p =< v_{k1}, v_{k2}, \ldots, v_{k|p|} >$ is a sequence of vertices, where $v_{ki} \in V$, $|p|$ indicating a number of vertices of p, wherein any two vertices within p are different. ■

In the present work, we did not consider paths in which a given vertex was visited more than once, because this would mean that the path contains a loop. For example, the path $< v_{11}, v_{12}, \ldots, v_{1n}, v, v_{21}, v_{22}, \ldots, v_{2n}, v, v_{31}, v_{32}, \ldots, v_{3n} >$ could be simplified as $< v_{11}, v_{12}, \ldots, v_{1n}, v, v_{31}, v_{32}, \ldots, v_{3n} >$.

Definition 6 (Edge Attributes of a Path): For a path $p =< v_{k1}, v_{k2}, \ldots, v_{k|p|} >$, the edge attributes of a path are $PEattr(p) = \{pw_1, pw_2, \ldots, pw_m\}$, where m is the number of edge attributes and pw_i can be evaluated using

$$pw_i = \sum_{j=1}^{|p|-1} w_{ji}, \tag{1}$$

where w_{ji} is the ith edge attribute of e_j and $e_j = (v_{kj}, v_{k(j+1)})$. ■

Definition 7 (Aggregate Attributes of a Path): For a path $p =< v_{k1}, v_{k2}, \ldots, v_{k|p|} >$, the aggregate attributes of a path are $PAattr(p) = \{pu_1, pu_2, \ldots, pu_n\}$, where n is the number of aggregate attributes and pu_i can be evaluated using

$$pu_i = \sum_{j=1}^{|p|} \sum_{k=1}^{|j|} u_{jk_i}. \tag{2}$$

■

Definition 8 (Path p_i Dominates Path p_j): Given two paths p_i and p_j, p_i is said to dominate path p_j if p_i is not worse than p_j in all edge attributes and aggregate attributes, and proves better in at least one edge attributes or aggregate attributes. ■

Definition 9 (Skyline Path Query With Aggregate Attributes): Given an undirected graph with start vertex s and end vertex t, a skyline path query with aggregate attributes returns all non-dominated paths from s to t. ■

IV. LOOKING FOR SKYLINE PATHS IN A ROAD NETWORK WITH AGGREGATE ATTRIBUTES

The most naïve approach to locating skyline paths in a road network with aggregate attributes is to identify all of the paths between the origin and destination in the network, calculate the edge and aggregate attributes of the paths, and perform a dominance check of all the attributes. Unfortunately, identifying all of the paths [15], [21] and calculating all of the edge and aggregate attributes can be extremely time-consuming. Thus, we developed an algorithm for locating skyline paths with aggregate attributes (SPA) without the need to identify all possible paths. The SPA algorithm uses a one-hop tree to divide the road network into multiple regions and then identify the regions through which the skyline paths cannot pass. Any paths passing through those regions are immediately eliminated, thereby reducing the search space. The concept of one-hop trees is introduced in the following.

A. ONE-HOP TREE

A one-hop tree is a height-balanced indexing structure capable of dividing road networks into multiple regions and then indexing them. Figure 5 presents an example of such a tree: Figure 5(a) presents the results of a road network using a partitioned one-hop tree, and Figure 5(b) is the corresponding one-hop tree. The vertices of the road network are stored in leaf nodes to form a leaf region, such as R_1 to R_5 in Figure 5(a). Multiple leaf nodes compose an internal node and multiple internal nodes form internal regions in the network. For example, in Figure 5(a), R_1 to R_3 make up R_6, while R_4 and R_5 constitute R_7. What is unique to this tree is that any vertex in any leaf region or internal region must be connected to other vertices in the same region. This is because the tree is based on the concept of single hops, which means that any two vertices (or regions) must be connected by an edge, such that only one hop is required to move from one vertex (or region) to another. All leaf regions R are formed as follows: R contains a center vertex c, and any vertices that are one hop away from c (in other words, connected to c by an edge) are included in R. Thus, even if vertex v_i in R is not connected by an edge to vertex v_j, it remains connected to v_j through c. This ensures that any vertex in R is directly or indirectly connected to all the other vertices. Note that an attempt is made to select vertices of a high degree to serve as the center vertex of leaf regions, where degree refers to the number of vertices to which it is connected. In Figure 5(a), R_2 is an example of a leaf region with its center, g, connected directly to h, i, and j. Thus, even though h, i, and j are not connected by edges, they are linked to one another through g. In internal region R', a child region R_c (where R_c may or may

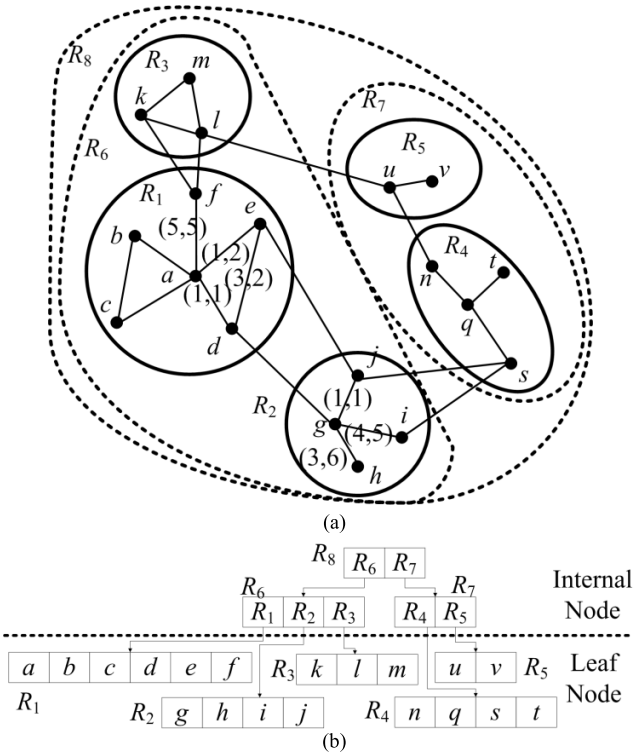

FIGURE 5. Example of one-hop tree: (a) Road network with one-hop tree; (b) Tree structure of the one-hop tree.

not be a leaf region) serves as the center, and any other child regions that are one hop away from R_c (i.e., connected to R_c by an edge) are included in R'. In this situation, despite the fact that child region R_x in R' is not connected to another child region R_y by an edge, R_x is nonetheless connected to R_y through R_c. Furthermore, all of the vertices in a given region are connected to one another, such that all of the vertices in different child regions are also linked. For example, R_6 in Figure 5(a) is an internal region with R_1 at its center. The three regions within R_6 (R_1, R_2, and R_3) are all connected to one another. In addition, any vertex in R_6 is connected to all other vertices in R_6.

1) PARAMETERS IN A LEAF NODE

Each leaf node in a one-hop tree contains seven parameters associated with the leaf region R_{leaf}. Four of these parameters are records concerning the vertices and edges associated with the leaf region and the other three are used by the SPA algorithm. The first four parameters are as follows: $Center(R_{leaf})$, $Vertex(R_{leaf})$, $Out(R_{leaf})$, and $Way(R_{leaf})$.

- $Center(R_{leaf})$ records the center of R_{leaf}.
- $Vertex(R_{leaf})$ lists all of the vertices in R_{leaf} that are connected by an edge to $Center(R_{leaf})$. Note that the number of vertices saved in $Vertex(R_{leaf})$ never exceeds the degree of $Center(R_{leaf})$ by more than one. This is because only the vertices connected by an edge to $Center(R_{leaf})$ are included in R_{leaf}, with the additional vertex referring to the $Center(R_{leaf})$ itself.
- $Out(R_{leaf})$ records the edges that connect R_{leaf} to other leaf regions. As shown in Figure 5(a), $Out(R_1)$ notes that R_1 is connected to R_8 through edges (e, j) and (d, g) and linked to R_3 through edges (f, k) and (f, l).
- $Way(R_{leaf})$ registers the paths that any given path p can take through R_{leaf}. For example, in Figure 5(a), path p can enter and leave R_3 by way of l and k, respectively, including $< l, m, k >$ and $< l, k >$. If p wishes to enter R_3 through l and leave through l, then the only alternative is $< l >$. Accordingly, $Way(R_3)$ would contain three paths: $< l, m, k >$, $< l, k >$, and $< l >$. Note that $< l, m, k, l >$ would not be included in $Way(R_3)$, because the definition of a path in this paper includes a stipulation that no path can pass through the same vertex twice.

In the following, we introduce the three parameters in the leaf region that are used in the SPA algorithm: $Min\_A(R_{leaf})$, $MPV\_E(R_{leaf})$, and $MPV\_A(R_{leaf})$.

- $Min\_A(R_{leaf})$ registers the minimum value of the aggregate attributes of the paths connecting any vertex in R_{leaf} to another vertex outside R_{leaf}. $Min\_A(R_{leaf})$ is a $1 \times n$ vector, where n denotes the number of aggregate attributes in the road network. For example, if R_5 in Figure 5(a) contained two vertices, u and v, then, $Min\_A(R_5)$ would equal $\min(Aattr(u, a), Aattr(u, b), \ldots, Aattr(u, t), Aattr(v, a), Aattr(vb), \ldots, Aattr(v, t))$.
- $MPV\_E(R_{leaf})$ is a $1 \times m$ vector, where m is the number of edge attributes in the road network, and "MPV_E" stands for minimum path value of edge attributes. This parameter records the minimum values of edge attributes obtained from all paths passing through R_{leaf}. For instance, Figure 5(a) includes six paths that pass through R_1: $< d, a, e >$, $< d, e >$, $< e, a, f >$, $< e, d, a, f >$, $< d, a, f >$, and $< d, e, a, f >$. Thus, $MPV\_E(R_1) = \min(PEattr(< d, a, e >), PEattr(< d, e >), PEattr(< e, a, f >), PEattr(< e, d, a, f >), PEattr(< d, a, f >), PEattr(< d, e, a, f >)) = \min((2, 3), (3, 2), (6, 7), (9, 8), (6, 6), (9, 9))=(2, 2)$. Note that if it were possible to use the same vertex v to enter and leave R_{leaf} , then all of the values in $MPV\_E(R_{leaf})$ would be 0. This is because all of the paths that pass through R_{leaf} must include $< v >$ and all of the associated edge attributes are 0. For example, the paths passing through R_3 in Figure 5(a) include $< l >$, with the result that all of the values in $MPV\_E(R_3)$ are 0.
- $MPV\_A(R_{leaf})$ is a $1 \times n$ vector, where n denotes the number of aggregate attributes in the road network, and "MPV_A" stands for minimum path value of the aggregate attributes. This parameter records the minimum values of aggregate attributes from all paths passing through R_{leaf}. The paths that pass through R_1 in Figure 5(a) include $< d, a, e >$, $< d, e >$, $< e, a, f >$, $< e, d, a, f >$, $< d, a, f >$, and $< d, e, a, f >$. Thus, $MPV\_A(R_1) = \min(PAattr(< d, a, e >), PAattr(< d, e >), PAattr(< e, a, f >), PAattr(< e, d, a, f >), PAattr(< d, a, f >), PAattr(< d, e, a, f >))$. Table 4 shows that this formula would result in $\min((13, 12), (4, 6), (16, 12), (30, 20), (11, 5), (30, 20))=(4, 5)$. Note that if it were possible to use the same vertex v to enter and leave R_{leaf}, then all of the values in $MPV\_A(R_{leaf})$ would be 0, for the same reason that was outlined previously.

Note that $MPV\_E(R_{leaf})$ and $MPV\_A(R_{leaf})$ are also values that must be included in the edge attributes and aggregate attributes of all paths p passing through R_{leaf}. Suppose that path p passed through R_1 in Figure 5(a). Then, path p would pass through one of the following six combinations: $< d, a, e >$, $< d, e >$, $< e, a, f >$, $< e, d, a, f >$, $< d, a, f >$, and $< d, e, a, f >$. Thus, the edge attribute of $p(PEattr(p))$ must include the minimum value of the edge attributes of these paths ($MPV\_E(R_1)$), and the aggregate attribute of p ($PAattr(p)$) must include the minimum value of the aggregate attributes of these paths ($MPV\_A(R_1)$).

2) PARAMETERS IN AN INTERNAL NODE

The internal nodes of a one-hop tree store seven parameters associated with internal region $R_{internal}$, four of which record the vertices and edges pertaining to the internal region with the other three used in the SPA algorithm. The first four are as follows: $Center(R_{internal})$, $Child(R_{internal})$, $Out(R_{internal})$, and $Way(R_{internal})$.

- $Center(R_{internal})$ registers the center of $R_{internal}$, which can be a leaf region or an internal region.
- $Child(R_{internal})$ records all of the child regions in $R_{internal}$, which must be connected to $Center(R_{internal})$ by

TABLE 4. Aggregate attributes between vertices in R1 and R2. (a) Aggregate attribute 1. (b) Aggregate attribute 2.

(a) AGGREGATE ATTRIBUTE 1

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
a	-									
b	2	-								
c	3	5	-							
d	1	7	6	-						
e	8	6	5	4	-					
f	1	1	6	9	7	-				
g	3	4	1	5	6	4	-			
h	1	2	6	2	2	4	5	-		
i	5	7	6	10	5	3	2	3	-	
j	6	3	9	4	4	4	1	6	7	-

(b) AGGREGATE ATTRIBUTE 2

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
a	-									
b	1	-								
c	5	3	-							
d	1	4	2	-						
e	5	7	9	6	-					
f	3	2	7	1	4	-				
g	1	3	2	6	5	4	-			
h	5	4	3	4	1	6	10	-		
i	2	2	7	4	3	1	5	4	-	
j	7	8	2	4	5	3	4	4	7	-

an edge. Note that the number of child regions stored in $Child(R_{internal})$ equals the degree of the $Center(R_{internal})$ plus one. This is because only the child regions connected to $Center(R_{internal})$ by an edge are included in r_f and the additional one refers to $Center(R_{internal})$ itself.

- $Out(R_{internal})$ registers the edges that connect $R_{internal}$ to other internal regions. As shown in Figure 5(a), $Out(R_6)$ indicates that R_6 is connected to R_7 through edges (l, u), (i, s), and (j, s).
- $Way(R_{internal})$ records the series of child regions that any given path p can take through $R_{internal}$. For the sake of convenience, we refer to a series of child regions as a combination of child regions (CCR). For example, in Figure 5(a), p has two alternative routes in its passage through R_6: (1) R_6 is entered through vertex l and exited through vertex i or j (i.e., p passes through R_3, R_1, and R_2 in R_6) and (2) R_6 is entered through i and exited through j (i.e., p only passes through R_2 in R_6). Thus, $Way(R_6)$ contains two CCRs: $< R_3, R_1, R_2 >$ and $< R_2 >$.

The three parameters in the internal region that are used in the SPA algorithm are as follows: $Min\_\ A(R_{internal})$, $MPV\_E(R_{internal})$, and $MPV\_A(R_{internal})$.

- $Min\_\ A(R_{internal})$ registers the minimum value of the aggregate attributes of paths connecting any vertex in $R_{internal}$ to other vertices outside $R_{internal}$; $Min\_\ A(RR_{internal})$ is a $1 \times n$ vector, where n denotes the number of aggregate attributes in the road network. If $R_{internal}$ contained child regions $R_{c1}, R_{c2}, \ldots, R_{cn}$, then, $Min\_\ A(R_{internal})$ would equal $\min(Min\_\ A(R_{c1}), Min\_\ A(R_{c2}), \ldots, Min\_A(R_{cn}))$.
- $MPV\_E(R_{internal})$ is a $1 \times m$ vector, where m represents the number of edge attributes in the road network. This parameter records the minimum values of the edge attributes in all paths associated with the various CCRs in $R_{internal}$. From $Way(R_{internal})$, we can derive that $R_{internal}$ contains k CCRs, ranging from CCR_1 to CCR_k. We can then write out CCR_i as $< R_{i1}, R_{i2}, \ldots, R_{iq} >$. Thus,

$$MPV\_E\ (R_{\text{internal}}) = \min_{i=1\sim k} \left(\sum_{j=1}^{q} MPV\_E\ (R_{ij}) \right), \tag{3}$$

where $\sum_{j=1}^{q} MPV\_E\ (R_{ij})$ is the sum of $MPV\_E(\bullet)$ from all child regions in CCR_i. For example, $Way(R_6)$ in Figure 5(a) includes two CCRs ($< R_3, R_1, R_2 >$ and $< R_2 >$), with the result that $MPV\_E(R_6)$ equals $\min((MPV\_E(R_3) + MPV\_E(R_1) + MPV\_E(R_2)), MPV\_E(R_2)) = \min(((0, 0) + (2, 2) + (1, 1)), (1, 1)) = \min((3, 3), (1, 1)) = (1, 1)$.

- $MPV\_A(R_{internal})$ is a $1 \times n$ vector, where n is the number of aggregate attributes in the road network. This parameter records the minimum values of all aggregate attributes from all paths associated with the various CCRs in $R_{internal}$. Suppose that $Way(R_{internal})$ contains k CCRs, ranging from CCR_1 to CCR_k. Then, CCR_i can be written as $< R_{i1}, R_{i2}, \ldots, R_{iq} >$. Thus,

$$MPV\_A\ (R_{\text{internal}}) = \min_{i=1\sim k} \left(\sum_{j=1}^{q} MPV\_A\ (R_{ij}) \right), \tag{4}$$

where $\sum_{j=1}^{q} MPV\_A\ (R_{ij})$ is the sum of $MPV\_A\ (\bullet)$ from all child regions in CCR_i. For example, $Way(R_6)$ in Figure 5(a) includes two CCRs ($< R_3, R_1, R_2 >$ and $< R_2 >$), such that $MPV\_A(R_6)$ equals $\min((MPV\_A(R_3) + MPV\_A(R_1) + MPV\_A(R_2)), MPV\_A(R_2))$. Furthermore, Table 4 shows that this formula results in $\min((0, 0)+(4, 5)+(1, 4), (1, 4)) = (1, 4)$; therefore, $MPV\_A(R_6) = (1, 4)$.

Note that $MPV\_E(R_{internal})$ and $MPV\_A(R_{internal})$ are values that must be included in the edge attribute and aggregate attribute for any path p passing through $R_{internal}$. Suppose that path p passes through R_6 in Figure 5(a). Then, the edge attribute of $p(PEattr(p))$ must include the minimum value of the edge attributes associated with these paths ($MPV\_E(R_6)$), and the aggregate attribute of $p(PAattr(p))$ must include the minimum value of the aggregate attributes associated with these paths ($MPV\_A(R_6)$).

3) TREE FORMULATION

The tree is created from the bottom up, starting from a leaf node and working upwards layer by layer until the first root node is established. In the formulation of leaf nodes, the algorithm first identifies the vertex with the highest degree, vertex v_1, and marks it as the center of the first leaf node $R_{leaf\,1}$. All of the vertices connected to v_1 through edges are added to $Vertex(R_{leaf\,1})$. Once $R_{leaf\,1}$ is established, the algorithm selects the vertex with the highest degree among the remaining vertices, vertex v_2, which is designated the

center of the second leaf node, leaf node $R_{leaf\,2}$. The vertices connected to v_2 by edges that have not been included in $Vertex(R_{leaf\,1})$ are added to $Vertex(R_{leaf\,2})$. Note that when the degrees of more than one vertex are the same, the algorithm processes the vertices according to the time they were listed, which does not affect the results. The method used for formulating the remaining leaf nodes is the same as that used for the second leaf node, and this process continues until all vertices in the road network have been added to leaf nodes. When all of the leaf nodes have been identified, we can calculate the five parameters for each: $Way(R_{leaf})$, $Out(R_{leaf})$, $Min\_A(R_{leaf})$, $MPV\_E(R_{leaf})$, and $MPV\_A(R_{leaf})$.

In the following, Figure 5(a) is used as an example to illustrate the establishment of leaf nodes. First, vertex a, the vertex with the highest degree in the road network (degree = 5) is designated the center of leaf node R_1. Then, b, c, d, e, and f, which are connected to a by edges, are included in $Vertex(R_1)$. The second step involves processing the vertex with the highest degree among the remaining vertices. Although the degrees of g and l are both 4, g was listed first and is therefore processed first. This means that g is designated as the center of R_2, whereupon h, i, and j (which are linked to g) are added to $Vertex(R_2)$. Note, that as long as $d$is connected to g, it is included in $Vertex(R_1)$ rather than being added to $Vertex(R_2)$. The method used for formulating R_3 to R_5 is the same as that used for R_2. Once all of the leaf nodes have been established, it is possible to calculate $Way(\bullet)$, $Out(\bullet)$, $Min\_A(\bullet)$, $MPV\_E(\bullet)$, and $MPV\_A(\bullet)$ for R_1 through R_5. Once completed, the results can be obtained as shown in Figure 5(a).

The formulation of internal nodes is outlined in the following. First, the algorithm identifies the leaf node with the highest degree, leaf node R_{c1}, and designates it the center of first internal node $R_{internal1}$. All other leaf nodes connected to R_{c1} by an edge are included in $Child(R_{internal1})$. Once $R_{internal1}$ is established, the leaf node with the next highest degree is classified as R_{c2} and designated the center of the second internal node $R_{internal2}$. The leaf nodes connected to R_{c2} by an edge and not included in $Child(R_{internal1})$ are added to $Child(R_{internal2})$. Note that when multiple leaf nodes have the same number of degrees, the algorithm selects the leaf node with the greatest number of vertices for processing, because eliminating this node would eliminate the greatest number of paths. The remaining internal nodes are formulated in the same manner as $R_{internal2}$, and the process continues until all of the leaf nodes have been added to internal nodes. Finally, when all of the internal nodes have been defined, we can calculate $Way(R_{internal})$, $Out(R_{internal})$, $Min\_A(R_{internal})$, $MPV\_E(R_{internal})$, and $MPV\_A(R_{internal})$ for each internal node.

In the following, we use Figure 5(a) as an example to illustrate the process of establishing internal nodes. First, the leaf node with the highest degree in the road network is identified and designated as the center of the first internal node. Although the degrees of leaf nodes from R_1 to R_5 are all 2, R_1 has a greater number of vertices and is therefore designated the center of internal node R_6. Then, R_2 and R_3, which are linked to R_1 by an edge, are added to R_6. The remaining leaf nodes R_4 and R_5 both have 2 degrees; however, R_4 contains a greater number of vertices and is therefore designated the center of internal node R_6. R_5 is connected to R_4 by an edge; therefore, it is added to R_6. When all of the leaf nodes have been added to internal nodes, we can calculate $Way(\bullet)$, $Out(\bullet)$, $Min\_A(\bullet)$, $MPV\_E(\bullet)$, and $MPV\_A(\bullet)$ for R_6 and R_6.

Even though the results may include a large number of internal nodes, the same method is applied to all of them to formulate internal nodes at a higher level. The process is continued until only a single node remains, which is the root node of the first level in the one-hop tree. For example, the root node in Figure 5(a) is R_8.

B. SPA ALGORITHM

Given origin s and destination t, the objective of the SPA algorithm is to identify the skyline paths between these two vertices without the need to search through every possible alternative. The SPA algorithm is based on the concept of sequential regions (SRs), which refers to a sequence of regions within the same layers of a one-hop tree, denoted as $< r_1, r_2, \ldots, r_n >$, in which edges connect r_1 and r_2, r_2 and $r_3, \ldots$, with r_{n-1} and r_n. Road networks can include multiple SRs connecting s and t(s and t are situated within r_1 and r_n in the SR, respectively), as well as SR α, which can include more than one path (paths $p_1, p_2, \ldots, p_m$) connecting s and t. The SPA algorithm calculates the minimum value of edge attributes, $MPV\_E(\alpha)$, from p_1 to p_m in α as well as the minimum value of aggregate attributes, $MPV\_A(\alpha)$, where MPV refers to the minimum path value, E denotes edge attributes, and A signifies aggregate attributes. If we identify any skyline path p_s in which $MPV\_E(\alpha)$ is dominated by $PEattr(p_s)$ and $MPV\_A(\alpha)$ is dominated by $PAattr(p_s)$, then we can conclude that paths p_1 to p_m in α cannot be skyline paths and eliminate them. In all other cases, the SPA algorithm assumes that paths p_1 to p_m in α are potential skyline paths, and thus expands α into multiple child SRs or paths. The SPA algorithm continues evaluating the candidacy of these child SRs or paths in becoming skyline paths until all possibilities have been explored.

The SPA algorithm requires a heap to maintain the SRs and paths as well as a list to store the skyline paths that have already been identified. Figure 6 exhibits the procedures preceding the implementation of the SPA algorithm, which is divided into an initialization phase and an examination phase.

1) INITIALIZATION PHASE

As shown in Figure 6, given origin s and destination t, the initialization phase comprises four actions: (1) identifying nodes r_s and r_t where s and t are located in the one-hop tree, (2) identifying all SRs connecting r_s and r_t, (3) calculating the minimum edge attribute and aggregate attribute values $MPV\_E(\bullet)$ and $MPV\_A(\bullet)$ of all paths in the identified SRs, and (4) inserting these SRs into the heap. The first action

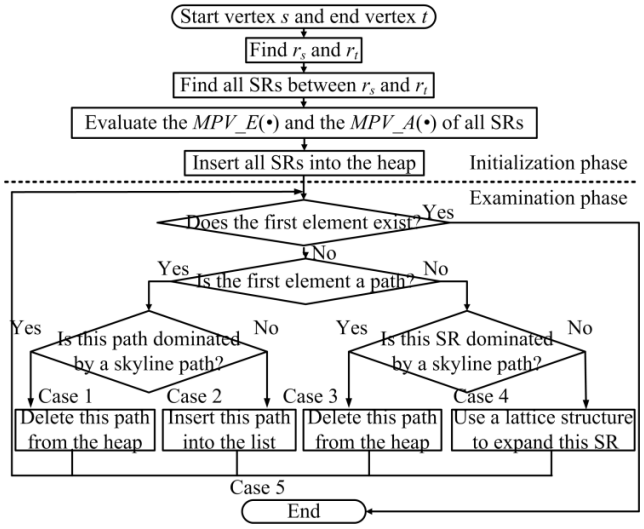

FIGURE 6. Flow chart of the SPA algorithm.

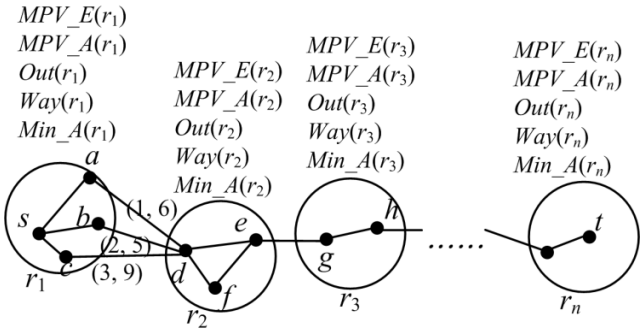

FIGURE 7. Example evaluation of MPV_E(α) and MPV_A(α).

begins with leaf nodes of the one-hop tree, which involves searching each layer from the bottom up for nodes r_s and r_t within the region of s and t. This is continued until r_s and r_t are contained within the same parent node. For example, let us assume that we want to find nodes in the region of vertices a and t in Figure 6(a). By checking all of the leaf nodes, we would determine that a and t are in R_1 and R_4, respectively. However, R_1 and R_4 are not included in the same parent node; therefore, we continue the checking process in the layer above the leaf node. After checking the internal node of this layer, it would be revealed that a and t are in R_6 and R_7, both of which are in root node R_8. The action would stop here with $r_s \in R_6$ and $r_t \in R_8$.

The second action involves identifying all of the SRs connecting r_s and r_t. Note that during the execution of this action, r_s and r_t are generally located in higher levels of the one-hop tree. Thus, most road networks contain fewer than 20 regions, which reduces the time required to locate SRs.

The third action involves calculating the values of minimum edge attributes and aggregate attributes, $MPV\_E(\bullet)$ and $MPV\_A(\bullet)$, of all paths in the SR. However, we must first elucidate the relationship between SRs and the paths within them. Figure 7 presents an example of SR α, $< r_1, r_2, \ldots, r_n >$ with the text adjacent to each region in the one-hop tree listing the corresponding parameters. Regions in the figure are interconnected by one or more edges, and r_1 and r_n include origin s and destination t, respectively. One path p must exist within α, starting from s in r_1, moving along one of the edges between r_1 and r_2, passing r_2, moving along one of the edges between r_2 and r_3, ..., moving along one of the edges between r_{n-1} and r_n, to reach t in r_n. In other words, p in α passes (1) region r_i ($2 \leq i \leq n-1$) and (2) the edge between r_i and r_{i+1} ($1 \leq i \leq n-1$). Thus, calculating the minimum edge attribute value of all paths in α requires that we consider two other values: (1) the addends that must be included in the edge attributes of p ($PEattr(p)$) when p passes r_i ($2 \leq i \leq n-1$) and (2) the addends that must be included in $PEattr(p)$ when p passes the edges between r_i and r_{i+1} ($1 \leq i \leq n-1$). The first value has the same value as the $MPV\_E(r_i)$ recorded in the one-hop tree, and therefore need not be re-calculated. Calculating the second value involves identifying edges $e_{i1}, e_{i2}, \ldots, e_{im}$, which are the edges connecting r_i and r_{i+1} in the $Out(r_i)$ of the one-hop tree. The use of $\min(Eattr(e_{i1}), Eattr(e_{i2}), \ldots, Eattr(e_{im}))$ makes it possible to determine the minimum value of the edge attribute, which is the value that must be included in $PEattr(p)$ (i.e., addend of $PEattr(p)$) when p passes the edges between r_i and r_{i+1}. In Figure 7, the three edges linking r_1 to r_2, have two edge attributes: (1, 6), (2, 5), (3, 9). Thus, we can conclude that when p passes from r_1 to r_2, $PEattr(p)$ must include $\min((1, 6), (2, 5), (3, 9)) = (1,5)$. Finally, from the two values above we derive the following:

$$MPV\_E\ (\alpha) = \sum_{i=2}^{n-1} MPV\_E\ (r_i) + \sum_{i=1}^{n-1} \min\ (Eattr\ (e_{i1})\ , Eattr\ (e_{i2})\ , ..., Eattr\ (e_{im})). \quad (5)$$

Note that the above formula does not consider $MPV\_E(r_1)$ or $MPV\_E(r_n)$ because they are values that must be included in $PEattr(p)$ when p passes through r_1 and r_n. However, p does not actually pass through r_1 and r_n in the SR, and therefore $MPV\_E(r_1)$ and $MPV\_E(r_n)$ cannot be directly included in the formula. As shown in Figure 7, p starts at point s in r_1 but does not actually pass all the way through r_1. Similarly, after entering r_n, p stops at t without passing all the way through r_n. It is possible to calculate the values that must be included in $PEattr(p)$ when p passes through r_1 and r_n; however, this would incur additional computation costs and should therefore be avoided.

Calculating the minimum aggregate attribute value of all paths in α, $MPV\_A(\alpha)$ requires that we also consider two values: (1) the addends that must be included in the aggregate attributes of p ($PAattr(p)$) when p passes r_i ($2 \leq i \leq n-1$) and (2) the addends that must be included in $PAattr(p)$ when p passes the edges in other regions r_i and r_j ($2 \leq i, j \leq n-1$, $i \neq j$). Again, we already know that the first value is the same as the one recorded for $MPV\_A(r_i)$ in the one-hop tree.

The computation process for the second value is more complex; therefore, Figure 7 is used to illustrate the values that must be included in *PAattr*(p) when p passes through different regions. Suppose that p only passes through r_2 and r_3 in Figure 6. From the one-hop tree, we can derive the minimum value of the aggregate attributes from any given vertex in r_2 to any other vertex outside of r_2, i.e., *Min_ A*(r_2). We can also derive the minimum value of the aggregate attributes from any given vertex in r_3 to any other vertex outside of r_3, i.e., *Min_ A*(r_3). As shown in Figure 7, when p passes through r_2, it passes vertices d and e; when passing through r_3, it passes vertices g and h. Consequently, when passing through r_2 and r_3, the addends of *PAattr*(p) must include the following four aggregate attribute values: *Aattr*(d, g), *Aattr*(d, h), *Aattr*(e, g), and *Aattr*(e, h). The minimum sum of these values is min(*Min_ A*(r_2), *Min_ A*(r_3)) $\times$ 2×2. As shown in this example, calculating the values that must be included in *PAattr*(p) when p passes through different regions in an SR requires (1) the minimum aggregate attribute values from any given vertex in r_i to any vertex outside of r_i, which is a value equal to *Min_ A*(r_i) in the one-hop tree, and (2) the number of vertices that p must pass on its way through r_i. The latter can be calculated using *Way*(r_i) of the one-hop tree and written as min$|Way(r_i)|$, where $|\bullet|$ denotes the number of vertices in the path. Accordingly, the second value is obtained as follows:

$$\sum_{i=2}^{n-1}\sum_{j=2}^{n-1}\left(\begin{array}{l}\min\left(Min\_A\left(r_i\right), Min\_A\left(r_j\right)\right)\\ \times \min\left|Way\left(r_i\right)\right| \times \min\left|Way\left(r_j\right)\right|\end{array}\right), \quad (6)$$

and *MPV_A*(α) equals

$$\sum_{i=2}^{n-1} MPV\_A\left(r_i\right) + \sum_{i=2}^{n-1}\sum_{j=2}^{n-1}\left(\begin{array}{l}\min\left(Min\_A\left(r_i\right), Min\_A\left(r_j\right)\right)\\ \times \min\left|Way\left(r_i\right)\right| \times \min\left|Way\left(r_j\right)\right|\end{array}\right), \quad (7)$$

The above formulas do not consider r_1 or r_n, due to the fact that p does not pass through r_1 or r_n in the SR; therefore, *MPV_A*($\bullet$), *Min_ A*($\bullet$), and *Way*($\bullet$) cannot be used directly. Of course, we can calculate the values that must be included in *PAattr*(p) when p passes through r_1 and r_n, but this would incur unnecessary computational costs.

The fourth action of the initialization phase involves adding the totals of the various dimensions of *MPV_E*($\bullet$) and *MPV_A*($\bullet$) from all SRs between s and t and inserting them in the heap in ascending order. The total number of dimensions serves as the basis for ranking, which is based on the fact that paths with smaller dimension totals are more likely to be skyline paths [6], [18]. Placing the SRs with the smaller total dimensions at the front of the heap makes it possible to obtain the skyline path results from the heap more quickly and eliminate the SRs and paths that do not qualify as skyline paths.

2) EXAMINATION PHASE

In the examination phase, the first element (which can be a path or an SR) is selected from the heap and a dominance check is performed using the skyline paths in the list. Each dominance check produces one of five outcomes as shown in Figure 6: (1) the first element is path p, which is dominated by one of the skyline paths in the list; (2) the first element is path p, which is not dominated by a skyline path in the list; (3) the first element is SR α in which *MPV_E*(α) and *MPV_A*(α) are dominated by *PEattr*(p_s) and *PAattr*(p_s) of skyline path p_s; (4) the first element is SR α in which *MPV_E*(α) and *MPV_A*(α) are not dominated by *PEattr*(p_s) and *PAattr*(p_s) of any skyline path p_s; or (5) the first element does not exist. The handling of these five outcomes proceeds as follows.

Case 1 (The First Element Is Path p, Which Is Dominated by a Skyline Path in the List): In this situation, p is dominated by another path and therefore cannot be a skyline path. It is eliminated from the heap.

Case 2 (The First Element Is Path p, Which Is Not Dominated by Any Skyline Path in the List): In this situation, as (1) p is not dominated by any skyline path, (2) the other possible paths in the heap cannot dominate p because their total dimensions exceed that of p [6], [18]. Thus, we surmise that no existing path can dominate p; i.e., it must be a skyline path and is added to the list.

Case 3 (The First Element Is SR α in Which MPV_E (α) and MPV_A (α) Are Dominated by PEattr (p_s) and PAattr (p_s) of Existing Skyline Path p_s): In this case, all of the paths in α are dominated by p_s; therefore, none of the paths in α could be skyline paths. Thus, we eliminate α from the heap.

Case 4 (The First Element Is SR α in Which MPV_E (α) and MPV_A (α) Are Not Dominated by PEattr (p_s) and PAattr (p_s) in Any Skyline Path p_s): In this situation, the paths in α are not dominated by any existing skyline paths; therefore, they may qualify as skyline paths themselves. Thus, we must identify all child SRs or paths in α (this method is discussed later), which are to be replaced in the heap according to their total dimensions for further evaluation by the algorithm.

Case 5 (The First Element Is a Blank): In this case, the algorithm has already checked all of the SRs and paths in the heap, such that the algorithm is terminated. The skyline paths registered in the list at this point become the final answers to the skyline path query.

The algorithm requires two steps to find all of the child SRs or paths in an SR $< r_1, r_2, \ldots, r_n >$. The first step involves identifying all possible CCRs or paths that could be taken in order to pass through the SR. The second step involves using a lattice structure to string these CCRs or paths into a child SR or a path linking the origin to the destination. Below, we describe these two steps in detail.

The first step depends on the type of region. To find the CCRs or paths that pass through r_2, r_3, ..., orr_{n-1} in SR $< r_1, r_2, \ldots, r_n >$, the algorithm only needs to read the *Child*($\bullet$) or *Way*($\bullet$) from r_2 to r_{n-1}, in which the CCRs and paths that pass through r_2, r_3, ..., or r_{n-1} were stored when

the one-hop tree was formulated. To find the CCRs or paths that pass through r_1 and r_n in SR $< r_1, r_2, \ldots, r_n >$, the algorithm is unable to use the *Child*(•) or *Way*(•) of r_1 and r_n because p does not actually pass through them. In this case, we can apply the brute force approach to identify the CCRs or paths that may be taken by p in r_1 and r_n. Note that the use of brute force algorithm in this instance does not waste very much time because the number of child regions or vertices stored in each region r in the one-hop tree is at most equal to the degree of *Center*(r) plus 1. Thus, even a brute force algorithm can quickly find all possible CCRs or paths in any region.

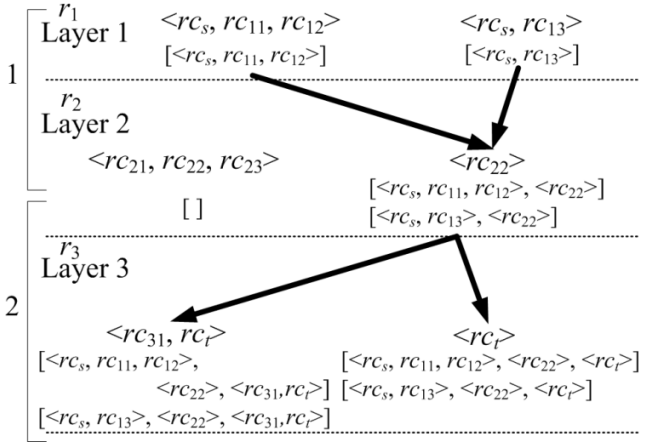

FIGURE 8. Lattice structure used in the SPA algorithm.

The second step involves the use of a lattice structure to string the CCRs or paths derived in the first step into a child SR or path connecting the origin to the destination. We first use the lattice structure in Figure 8 to identify all of the child SRs in SR $< r_1, r_2, r_3 >$. The lattice structures of all paths in the SR can be derived from this figure by analogy. Layer i of the lattice structure records all of the CCRs in region i in the SR. In Figure 8, Layer 1 registers CCRs $<rc_s, rc_{11}, rc_{12} >$ and $<rc_s, rc_{13} >$ in r_1, where rc_s is the child region in which origin s is located. Layer 2 records CCRs $<rc_{21}, rc_{22}, rc_{23} >$ and $<rc_{22} >$ in r_2. Layer 3 contains CCRs $<rc_{31}, rc_t >$ and $<rc_t >$ in r_3, where rc_t is the child region in which destination t is located. The number of edges between layers is also recorded in the lattice structure. On the left of Figure 8, we can see a number 1 between r_1 and r_2, which indicates that there is one edge connecting them. The number 2 between r_2 and r_3 indicates that there are two edges connecting them. The CCRs of different layers are also linked by arrows, indicating the presence of an edge connecting them. For example, $<rc_s, rc_{13} >$ in Layer 1 and $<rc_{22} >$ in Layer 2 are connected by an arrow, which means that an edge connects $<rc_s, rc_{13} >$ and $<rc_{22} >$. Figure 8 also lists combinations of CCRs as an indication of the paths that can be taken from rc_s to reach the selected CCR. For example, we can see $[<rc_s, rc_{11}, rc_{12} >, <rc_{22} >]$ and $[<rc_s, rc_{13} >, <rc_{22} >]$ below $<rc_{22} >$ in Layer 2, which means that rc_s can reach $<rc_{22} >$ through rc_{11}, rc_{12}, or rc_{13}. Furthermore, [ ] beneath a CCR indicates that rc_s is unable to reach this CCR. For example, [ ] beneath $<rc_{21}, rc_{22}, rc_{23} >$ in Layer 2 indicated that rc_s is unable to reach $<rc_{21}, rc_{22}, rc_{23} >$.

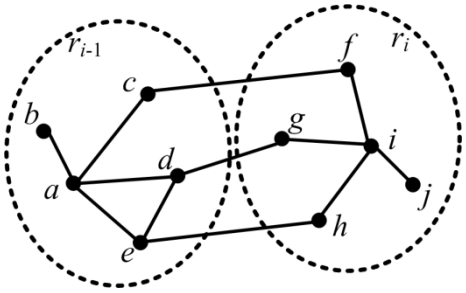

FIGURE 9. Example of a path running back and forth between two layers.

To identify the child SRs, we start from the first layer of the lattice structure and progress through them layer by layer. Each layer requires three actions. Suppose that the layer currently being processed is Layer i. The three actions would be as follows: (1) Identify the CCRs in Layer $i-1$ that are connected to CCRs in Layer i by an edge, and link the two in the lattice structure. (2) Check whether a path can run back and forth between CCRs in Layer $i-1$ and CCRs in Layer i, and take this into consideration in the lattice structure as well. Running back and forth would require the existence of at least three edges between the two layers. This would enable a path to run from Layer $i-1$ to Layer i and then from Layer i to Layer $i-1$, and finally from Layer $i-1$ to Layer i again. Figure 9 presents an example of a path running back and forth between different layers. As can be seen, r_{i-1} reaches r_i via edge (c, f), whereupon r_i goes back to r_{i-1} via edge (g, d) and finally returns to r_i via edge (e, h). In other words, the addition of paths to $< r_{i-1}, r_i >$ in Figure 9 requires that we take into account how the paths run in $< r_{i-1}, r_i, r_{i-1}, r_i >$ and add these situations to the lattice structure. (3) For each CCR combination β in Layer i, we calculate the addends of the edge attributes and aggregate attributes of path p passing through this CCR combination, namely $MPV\_E(\beta)$ and $MPV\_A(\beta)$. We then check whether $MPV\_E(\beta)$ and $MPV\_A(\beta)$ are dominated by the $PEattr(p_s)$ and $PAattr(p_s)$ of skyline path p_s. Note that the methods used to calculate $MPV\_E(\beta)$ and $MPV\_A(\beta)$ in this step are similar to those used for the $MPV\_E(\bullet)$ and $MPV\_A(\bullet)$ of an SR. This is because the calculation objectives of the two are the same, and CCRs and SRs both consist of multiple regions on the same layer of a one-hop tree. The only difference is that an SR considers regions r_s and r_t in which the origin and destination are located, while a CCR only considers region r_s where the origin is located. Suppose that CCR β is $< r_1, r_2, \ldots, r_n >$, then r_1 contains origin s, and edges e_{i1}, $e_{i2}, \ldots$, and e_{im} connects r_i and r_{i+1}. From Equation (5), we determine $MPV\_E(\beta)$ as follows:

$$MPV\_E(\beta) = \sum_{i=2}^{n} MPV\_E(r_i) + \sum_{i=1}^{n-1} \min\left(Eattr(e_{i1}), Eattr(e_{i2}), \ldots, Eattr(e_{im})\right), \quad (8)$$

where n-1 in Equation (5), $\sum_{i=2}^{n-1} MPV\_E\,(r_i)$, becomes n in Equation (8). This is because the last region r_n in a CCR does not include final destination t as the last region. Even though a path does not pass through the final region in an SR, it does pass through the final region of a CCR; therefore, we must consider r_n in Equation (8). Similarly, supposing that CCR β is $< r_1, r_2, \ldots, r_n >$, then Equation (7) would render $MPV\_A(\beta)$ as follows:

$$MPV\_A\,(\beta) = \sum_{i=2}^{n} MPV\_A\,(r_i) + \sum_{i=2}^{n}\sum_{j=2}^{n}\left(\begin{array}{l}\min\left(Min\_A\,(r_i)\,, Min\_A\left(r_j\right)\right) \\ \times \min\left|Way\,(r_i)\right| \times \min\left|Way\left(r_j\right)\right|\end{array}\right), \quad (9)$$

where $n-1$ in Equation (7) becomes n in Equation (9). The reason is the same as that in Equation (8); i.e., even though a path does not pass through the final region in an SR, it does pass through the final region of a CCR and therefore, r_n must be considered in Equation (9).

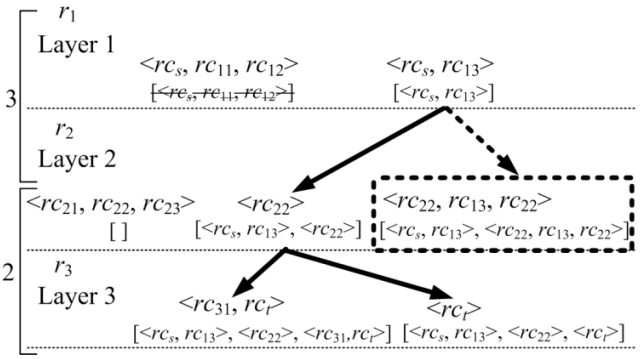

FIGURE 10. Example of using the lattice structure to find child SRs.

In the following, Figure 10 is used to explain the methods involved in identifying all child SRs in SR $< r_1, r_2, r_3 >$ using a lattice structure with three edges connecting r_1 and r_2 and two edges connecting r_2 and r_3. Note that all of the paths in the lattice structure can be derived using this simple example. We begin by processing Layer 1 of the lattice structure. Note that this layer does not require the first or second actions because it is the first layer. The third action involves calculating the $MPV\_E(\bullet)$ and $MPV\_A(\bullet)$ of $[<rc_s, rc_{11}, rc_{12}>]$ and $[<rc_s, rc_{13}>]$ using Equations (8) and (9), respectively. Suppose that the results indicate that $MPV\_E(<rc_s, rc_{11}, rc_{12}>)$ is dominated by the $PEattr(p_s)$ of skyline path p_s, and $MPV\_A(<rc_s, rc_{11}, rc_{12}>)$ is dominated by PAattr(p_s). If none of the paths in $< rc_s, rc_{11}, rc_{12} >$ can become a skyline path, we can draw a line through $[<rc_s, rc_{11}, rc_{12}>]$ in Figure 10, which eliminates the paths in this branch from further consideration.

The first step in processing Layer 2 in the lattice structure involves identifying the CCRs connected by edges between Layers 1 and 2. The algorithm first finds the last child region rc_{13} of the remaining CCR $<rc_s, rc_{13}>$ in Layer 1 as well as the first child regions of CCRs $<rc_{21}, rc_{22}, rc_{23}>$ and $<rc_{22}>$ in Layer 2: rc_{21} and rc_{22}. Suppose that $Out(rc_{13})$ in the one-hop tree indicates that rc_{13} is connected to rc_{22} but not to rc_{21}. We would then connect $<rc_s, rc_{13}>$ to $<rc_{22}>$ in Figure 10 but not $<rc_s, rc_{13}>$ to $<rc_{21}, rc_{22}, rc_{23}>$. This would be the same under $<rc_{21}, rc_{22}, rc_{23}>$ [ ]. The second step involves checking whether any paths can run back and forth between the CCRs in Layer 1 or those in Layer 2. More than three edges connect Layers 1 and 2; therefore, there may be paths running back and forth between the CCRs in Layer 1 and/or the CCRs in Layer 2. Furthermore, the results of the first action tell us that rc_{13} and rc_{22} share the only connection between Layer 1 and Layer 2. The algorithm then utilizes the brute force approach to determine whether any paths run back and forth between rc_{13} and rc_{22}. If any were identified, then we would add CCR $<rc_{22}, rc_{13}, rc_{22}>$ to Layer 2 in Figure 10 in addition to the original $<rc_{22}>$ and connect $<rc_{22}, rc_{13}, rc_{22}>$ to $<rc_s, rc_{13}>$ using a dashed line. The third action in Layer 2 involves calculating the $MPV\_E(\bullet)$ and $MPV\_A(\bullet)$ of $[<rc_s, rc_{13}>, <rc_{22}>]$ and $[<rc_s, rc_{13}>, <rc_{22}, rc_{13}, rc_{22}>]$ using Equations (8) and (9), respectively. Doing so indicates that a skyline path dominates the CCR combination $[<rc_s, rc_{13}>, <rc_{22}, rc_{13}, rc_{22}>]$; therefore, we draw a line through this CCR combination to indicate that it has been eliminated.

The first step in Layer 3 involves identifying the CCRs connected by edges in Layers 2 and 3. In this case, rc_{31} and rc_t are connected to rc_{22}; therefore, we draw lines linking $<rc_{22}>$ to $<rc_{31}, rc_t>$ and $<rc_t>$. The second step involves checking whether any paths run back and forth between Layer 2 and Layer 3. As shown in Figure 9, only two edges connect the two layers; therefore, this is not possible and no action is required. The third action involves calculating the $MPV\_E(\bullet)$ and $MPV\_A(\bullet)$ of $[<rc_s, rc_{13}>, <rc_{22}>, <rc_{31}, rc_t>]$ and $[<rc_s, rc_{13}>, <rc_{22}>, <rc_t>]$ using Equations (8) and (9). The results fail to identify any skyline paths dominating the two CCR combinations; therefore, they are retained in the lattice structure. This is the last layer of the lattice structure; therefore, $<rc_s, rc_{13}, rc_{22}, rc_{31}, rc_t>$ and $<rc_s, rc_{13}, rc_{22}, rc_t>$ are the child SRs of SR $< r_1, r_2, r_3 >$.

C. TEMPORAL AND SPATIAL COMPLEXITY

In this section, we examine the performance of the proposed method from the perspective of temporal and spatial complexity. Generally speaking, the ideal situation for a skyline-related query would involve only one result in the dataset. Thus, using the proposed method to obtain the final result for a given query would involve traversing the one-hop tree from root to leaf only once. If the height of the one-hop tree were h, then the temporal complexity would be O(h). Thus, the spatial complexity would be O($h * l$), where l is the size of the space used by each node in a hop-tree.

In contrast, the worst case for a skyline-related query would be a situation in which most of the points in the dataset are answers. In this work, that would be a situation in which most of paths between the origin and destination end up as results for the query. In other words, even if all of the nodes

and edges in the road network were pre-indexed, they would still have to be accessed, such that the temporal complexity would be equal to that of the naïve method. Furthermore, all of the nodes in the hop-tree would have to be expanded, but the proposed method would be unable to take advantage of the naïve method. This lends credence to the argument proposed by Sheng and Tao [19] that the complexity of all index-based skyline methods necessarily equals to that of the naïve method. Nonetheless, the fact that worst-case situations are seldom encountered means that the proposed method is still a viable solution.

V. SIMULATIONS

In the following section, we describe a series of simulations used to evaluate the efficiency with which skyline paths can be found using the SPA algorithm with one-hop tree. This is the first study to seek skyline paths from a road network with aggregate attributes; therefore, we also examined the influence of aggregate attributes and edge attributes on skyline paths.

TABLE 5. A summary of simulation parameters.

Parameter	Values
Number of vertices	200, 300, **400**, 500, 600
Maximum degree of vertices	4, **8**, 12, 16, 20
(N_E, N_A)	(1, 2), **(2, 2)**, (3, 2), (4, 2), (5, 2), (2, 1), (2, 3), (2, 4), (2, 5)

The dataset used in this simulation was a bus network generated from the road network of Oldenburg, Germany, which is a benchmark road network comprising 6,105 intersections and 7,036 road segments [2], [7], [10]. Simulating a real-world bus network involved the random selection of intersections on the Oldenburg road network to serve as bus stops (represented as vertices). Bus stops were randomly connected to a number of adjacent bus stops (represented as edges). Following the construction of the bus network, the values of edge attributes and aggregate attributes were randomly generated within a range 0 to 1. Table 5 summarizes the three related parameters, the default parameters of which are listed in bold type. The number of vertices corresponds to the number of bus stops and the maximum degree of the vertices corresponds to the maximum number of bus stops that could be connected to a given bus stop. N_E and N_A indicate the number of edge attributes and aggregate attributes, respectively. These were set to ensure that every scenario included at least two edge attributes or at least two aggregate attributes. To mediate the influence of different road networks and different sets of origin and destination points, the final results were averaged from 900 experiments. We began by generating 30 bus networks on the Oldenburg road network and then randomly assigning 30 sets of origin and destination points for each bus network. All of the experiments were performed on a computer running Microsoft Windows XP on an Intel i7-3770 CPU at 3.40GHz with 4 GB of main memory. All programs were written in MATLAB®.

A. INFLUENCE OF AGGREGATE ATTRIBUTES AND EDGE ATTRIBUTES ON SKYLINE PATHS

We used the average number and average length of skyline paths to analyze the influence of aggregate and edge attributes on skyline paths. Note that the average length of skyline paths indicates the average number of vertices included in a skyline path.

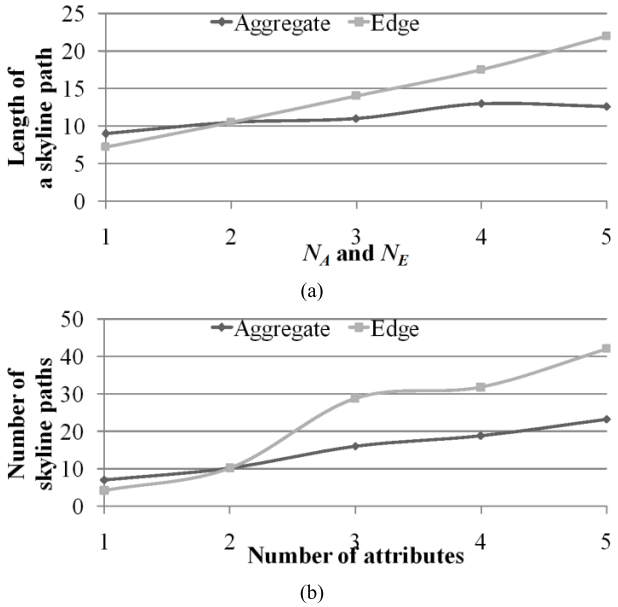

FIGURE 11. Simulation results from varying N_A and N_E for (a) the average length of a skyline path, (b) the average number of skyline paths. In both graphs, the "aggregate" data have the number of "edge attributes" fixed at 2, and likewise for the "edge" data.

Figure 11 shows how the values of N_A and N_E affected the average length and average number of skyline paths. Note that for the aggregate curve in Figure 11, the number of aggregate attributes varied from one to five and the number of edge attributes was fixed at 2. For the edge curve, the number of the edge attributes varied from one to five and the number of aggregate attributes was fixed at 2. Figure 11(a) indicates that the average length of skyline paths increased with an increase in $N_A + N_E$. This is because the aggregate attribute values and the edge attribute values of path p both increased with the length of p. Thus, when $N_A + N_E$ is small, a shorter path is more likely to dominate a longer path, such that most of the skyline paths end up being quite short, which reduces the average length of the skyline paths. However, when $N_A + N_E$ is larger, the likelihood that a shorter path will dominate a longer path decreases, with the result that longer paths have a greater chance of being selected and the average length of the skyline path is extended. Figure 11(a) also shows that compared to an increase in N_E, an increase in N_A would produce a slower increase in the average length of the skyline paths. This is because the values of aggregate attributes present a pronounced increase with the length of the path; however, the values of edge attributes present only a slight increase. For example, when a new vertex is added

to path p with length n, the aggregate attribute values of p need to add additional n values. However, only one value needs to be added to the edge attribute values of p. Under these conditions, the values of aggregate attributes for p with a longer path are far greater than for p' with a shorter path. Thus, even when N_A is increased, p is still likely to be dominated by p', thereby reducing its chance of being a skyline path, which reduces the average length of the skyline paths. In contrast, increases in the value of edge attributes due to an increase in the length of the path are not as pronounced as similar increases in aggregate attributes, with the result that longer paths still have a chance of being a skyline path when N_E is increased, ultimately increasing the average length of the skyline paths.

Figure 11(b) presents the average number of skyline paths in the road network with various values for N_E and N_A. The number of skyline paths increased with an increase in the number of either type of attribute. This is because when $N_A + N_E$ increases, longer paths are more likely to be selected as skyline paths, thereby increasing the number of skyline paths. Figure 11(b) also indicates that, compared to an increase in N_E, an increase in N_A results in a less pronounced increase in the average number of skyline paths. This is because longer paths do not become skyline paths, even if N_A is increased, with the result that the number of skyline paths grows more slowly. In contrast, increasing N_E increases the likelihood that a longer path will become a skyline path, such that the number of skyline paths grows more rapidly. The above analysis demonstrates that the influence of N_E on skyline paths is more pronounced than that of N_A.

B. EFFICIENCY OF SPA ALGORITHM WITH ONE-HOP TREE

In this section, we compare the performance of the SPA algorithm in conjunction with the one-hop tree (SPA-one) to the SPA algorithm with the R-tree (SPA-R). Other skyline path algorithms were not included in this analysis because existing skyline path algorithms are unable to find skyline paths when aggregate attributes are present in the road network. The brute-force algorithm was also excluded because it must identify all paths between the origin and destination points, calculate the edge and aggregate attributes, and perform dominance checks for the paths, making it a highly inefficient method for finding skyline paths in road networks with aggregate attributes. Three experiments were conducted to investigate how the performance of SPA-one would be affected by (N_E, N_A), the number of vertices, and the maximum number of degrees for each vertex.

Figure 12 lists the average number of paths that were checked and the time required by SPA-one and SPA-R with N_E and N_A set to various values. Note that for the "SPA-one (A)" and the "SPA-R (A)" curves in Figure 12, the number of aggregate attributes varies between one and five, whereas the number of edge attributes is fixed at 2. For the "SPA-one (E)" and "SPA-R (E)" curves, the number of the edge attributes varies between one and five, whereas the number of aggregate attributes is fixed at 2. In each of the

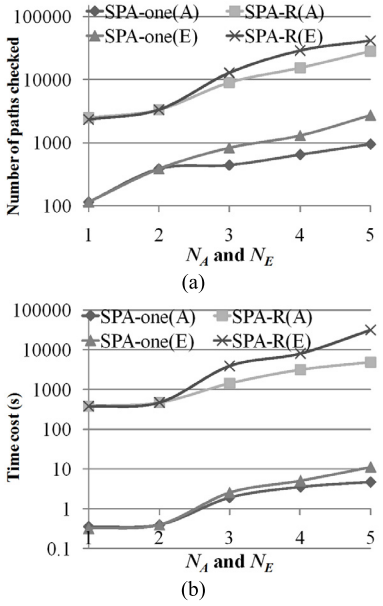

FIGURE 12. Simulation results for various values of N_A and N_E, showing for SPA-one and SPA-R, (a) number of paths checked, (b) time cost.

diagrams, a logarithmic scale was applied to the y-axis to depict several orders of magnitude. As shown in Figure 12, the number of paths that were checked and the time required increased with the number of attributes, regardless of which attributes were changed and which algorithm was used. This can be attributed to an increase in the number of skyline paths following an increase in the number of attributes. Figure 12 also shows that the number of paths that were checked and the time required for SPA-one were at least an order of magnitude lower than for SPA-R. This is because the one-hop tree attempts to involve the greatest number of vertices and viable paths for each region. By determining that no skyline path enters a particular region, the algorithm greatly reduces the number of paths to be checked and the subsequent time required for calculation. Furthermore, the definition of an R-tree does not require that any two vertices within a given region be connected, which means that a region may include only a few or perhaps no viable paths. Thus, even if these regions are eventually removed by the algorithm, the number of paths and degree to which computation time can be reduced using SPA-R is limited. In short, SPA-R needs to check more paths and therefore requires more time.

Figure 12 also indicates that regardless of whether SPA-one or SPA-R were used, an increase in N_E would result in a greater number of paths and longer computation time, compared to an increase in N_A. This is because N_E is able to produce a greater number of skyline paths, as shown in Figure 11. Finally, Figure 12(a) shows that regardless of which attribute was increased, SPA-R checked 10 times as many paths as did SPA-one, and required 100 times as much time, as shown in Figure 12(b). This can be attributed to the fact that SPA-R needs to reconfigure a path whenever it

discovers vertices at the boundary of a particular region that are not connected within that region. In contrast, the one-hop tree ensures that any two vertices within a given region are connected by at least one path that lies entirely within that region.

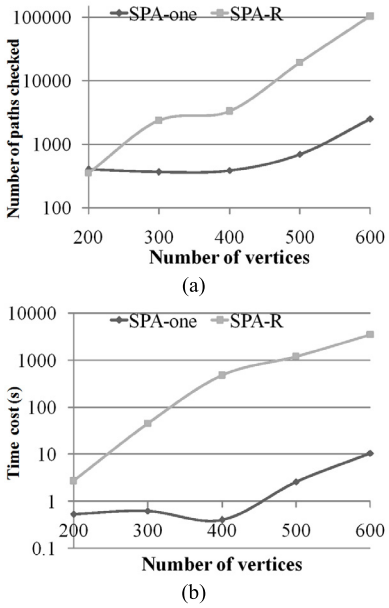

FIGURE 13. Simulation results for various numbers of vertices, showing for SPA-one and SPA-R, (a) number of paths checked, (b) time cost.

Figure 13 lists the average number of paths checked and the time required by SPA-one and SPA-R using various numbers of vertices. Again, we applied a logarithmic scale to the y-axes of the two diagrams to depict several orders of magnitude. Figure 13 clearly shows that regardless of which algorithm was used, an increase in the number of vertices resulted in an increase in the number of paths that were checked as well as an increase in the time required for computation. This is because increasing the number of vertices increases the number of possible paths. Figure 13 also shows that the results were much lower for SPA-one than for SPA-R. As shown in Figure 12, each region of a one-hop tree contains a greater number of vertices and paths, compared to the regions associated with an R-tree. As a result, SPA-one eliminates a greater number of paths and saves more time. Figure 13 illustrates how the disparity between SPA-one and SPA-R expands with the number of vertices. This is because a small number of vertices limits the number of paths in a road network, which also limits the number of paths that could be eliminated, regardless of whether SPA-one or SPA-R were used. Hence, differences between the two algorithms are less pronounced. Increasing the number of vertices increases the number of paths in the road network that could be eliminated by SPA-one. This clearly illustrates the reasons for the immense differences between the two algorithms with regard to the number of paths to be checked and the time required for computation.

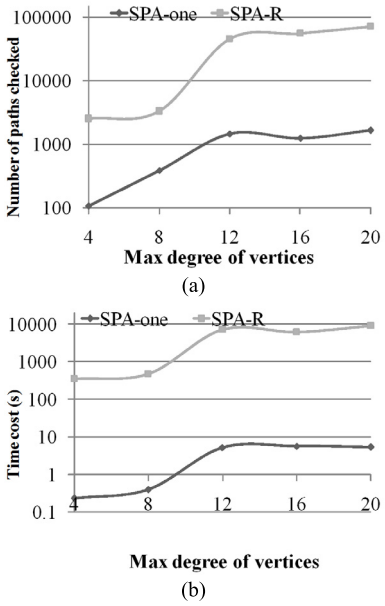

FIGURE 14. Simulation results for various maximum degrees of vertices, showing for SPA-one and SPA-R, (a) number of paths checked, (b) time cost.

Figure 14 presents the average number of paths that are checked and the time required by SPA-one and SPA-R using various values for the maximum degree of vertices. A logarithmic scale was applied to the y-axes of the two diagrams. When the maximum degree was lower than 12, all four curves grew until they reached 12, whereupon they leveled off. This is because an increase in degree from 4 to 12 increases the number of viable paths in the road network by increasing the number of bus stops to which each bus stop could be connected. This would increase the number of potential paths and skyline paths that would have to be checked between the origin and destination. However, increasing the degree from 12 to 20 would probably not increase the number of skyline paths, even if a new connectible bus stop v' were added to bus stop v. When a bus network is being established, each bus stop with first priority connects to adjacent bus stops before connecting to more distant stops. When the degree is greater than 12, the newly connected bus stops are those located furthest away. In the event that a bus travels from v (the original bus stop) to v' (a newly-connected distant bus stop), a detour might be required to follow a path that is longer. However, as discussed in Section A of this chapter, a longer bus path is less likely to become a skyline path. Even if v and v' are connected, the number of skyline paths in the road network will not increase, such that the paths that need to be checked and time required by the algorithm will not increase either.

VI. CONCLUSION

This paper presents a novel query method for the identification of skyline paths in cases involving aggregate attributes in a road network. We began by outlining the difficulties involved in identifying skyline paths when aggregate

attributes are present in road networks, as well as the reasons for the inability of existing skyline path algorithms to overcome them. This study presented a one-hop tree and SPA algorithm to overcome this limitation, and simulation results demonstrate the efficacy of this approach.

The value of this study lies in its wide-scale applicability. This work could be extended to support what is referred to as skyline network queries. For example, the manager of a metro company might seek to develop a new metro system comprising several metro routes on a road network with aggregate attributes. In such cases, the algorithm developed in this study would be limited in the following two respects: (1) The proposed algorithm would be able to find only one skyline path at a time, such that the selection of multiple skyline paths would require running the algorithm multiple times. (2) The fact that each vertex in the metro network can connect with any other vertex would increase the complexity involved in calculating edge and aggregate attributes. Thus, performing a skyline network query on a real-world road network would require an algorithm capable of finding multiple skyline paths. The calculation of edge and aggregate attributes could be accelerated through the use of pruning methods to reduce the number of vertices and edges considered by the algorithm. The developments in this paper represent a major advancement in the application of skyline queries to networks in the real world.

REFERENCES

[1] S. Aljubayrin, Z. He, and R. Zhang, "Skyline trips of multiple POIs categories," in *Proc. Int. Conf. Database Syst. Adv. Appl. (DASFAA)*, 2015, pp. 189–206.
[2] I. Bartolini, P. Ciaccia, and M. Patella, "SaLSa: Computing the skyline without scanning the whole sky," in *Proc. Int. Conf. Inf. Knowl. Manage. (CIKM)*, 2006, pp. 405–414.
[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proc. ACM Special Interest Group Manage. Data (SIGMOD)*, 1990, pp. 322–331.
[4] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. IEEE 17th Int. Conf. Data Eng. (ICDE)*, Apr. 2001, pp. 421–430.
[5] Z. Chen, H. T. Shen, X. Zhou, and J. X. Yu, "Monitoring path nearest neighbor in road networks," in *Proc. ACM Special Interest Group Manage. Data (SIGMOD)*, 2009, pp. 591–602.
[6] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, Mar. 2003, pp. 717–719.
[7] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *Proc. Int. Conf. Vary Large Data Bases (VLDB)*, 1997, pp. 426–435.
[8] K. Deng, Y. Zhou, and H. Tao, "Multi-source skyline query processing in road networks," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, Apr. 2007, pp. 796–805.
[9] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *SIGMOD Rec.*, vol. 14, no. 2, pp. 47–57, 1984.
[10] W. T. Hsu, Y. T. Wen, L. Y. Wei, and W. C. Peng, "Skyline travel routes: Exploring skyline for trip planning," in *Proc. IEEE Int. Conf. Mobile Data Manage. (MDM)*, Jul. 2014, pp. 31–36.
[11] Y.-K. Huang, C.-H. Chang, and C. Lee, "Continuous distance-based skyline queries in road networks," *Inf. Syst.*, vol. 37, no. 7, pp. 611–633, 2012.
[12] Y.-K. Huang and Z.-H. He, "Processing continuous K-nearest skyline query with uncertainty in spatio-temporal databases," *J. Intell. Inf. Syst.*, vol. 45, no. 2, pp. 165–186, 2015.
[13] X. Huang and C. S. Jensen, "In-route skyline querying for location-based services," in *Proc. Web Wireless Geograph. Inf. Syst. (W2GIS)*, 2004, pp. 120–135.
[14] S. Jang and J. Yoo, "Processing continuous skyline queries in road networks," in *Proc. Int. Symp. Comput. Sci. Appl.*, Oct. 2008, pp. 353–356.
[15] H.-P. Kriegel, M. Renz, and M. Schubert, "Route skyline queries: A multi-preference path planning approach," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, Mar. 2010, pp. 261–272.
[16] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng, "On trip planning queries in spatial databases," in *Proc. Int. Symp. Spatial Temporal Databases (SSTD)*, 2005, pp. 273–290.
[17] S. Pan, Y. Dong, J. Cao, and K. Chen, "Continuous probabilistic skyline queries for uncertain moving objects in road network," *Int. J. Distrib. Sensor Netw.*, vol. 2014, Mar. 2014, Art. no. 365064.
[18] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proc. ACM Special Interest Group Manage. Data (SIGMOD)*, 2003, pp. 467–478.
[19] C. Sheng and Y. Tao, "Worst-case I/O-efficient skyline algorithms," *ACM Trans. Database Syst.*, vol. 37, no. 4, 2012, Art. no. 26.
[20] W. Son, S.-W. Hwang, and H.-K. Ahn, "MSSQ: Manhattan spatial skyline queries," *Inf. Syst.*, vol. 40, pp. 67–83, Mar. 2014.
[21] Y. Tian, K. C. K. Lee, and W.-C. Lee, "Finding skyline paths in road networks," in *Proc. ACM Int. Conf. Adv. Geograph. Inf. Syst. (SIGSPATIAL)*, 2009, pp. 444–447.
[22] Y.-T. Wen, K.-J. Cho, W.-C. Peng, J. Yeo, and S.-W. Hwang, "KSTR: Keyword-aware skyline travel route recommendation," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2015, pp. 449–458.
[23] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang, "Stochastic skyline route planning under time-varying uncertainty," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, Mar./Apr. 2014, pp. 136–147.
[24] *Bing Maps*, accessed on Aug. 8, 2016. [Online]. Available: http://www.bing.com/maps/
[25] *Google Maps*, accessed on Aug. 8, 2016. [Online]. Available: https://maps.google.com/
[26] *MapQuest*, accessed on Aug. 8, 2016. [Online]. Available: http://www.mapquest.com/
[27] *Yahoo! Maps*, accessed on Aug. 8, 2016. [Online]. Available: https://maps.yahoo.com/

YI-CHUNG CHEN (M'16) received the B.S. and M.S. degrees in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 2007 and 2008, respectively, and the Ph.D. degree from the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, in 2014. He is currently an Assistant Professor with the Department of Information Engineering and Computer Science, Feng Chia University. His research interests include databases, recommendation system, social network analysis, and artificial intelligences.

CHIANG LEE received the B.S. degree from National Cheng-Kung University, Taiwan, in 1980, and the M.E. and Ph.D. degrees in electrical engineering from the University of Florida, Gainesville, in 1986 and 1989, respectively. He joined IBM Mid-Hudson Laboratories, Kingston, NY, USA, in 1989 and participated in a project involved in the design and performance analysis of a parallel and distributed database systems. He joined the Faculty of National Cheng Kung University in 1990 and is currently a Professor with the Department of Computer Science and Information Engineering. He has authored or co-authored many papers in major journals and conferences, and has been invited as an author of a chapter for several technical books.