JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 36, 921-935 (2020)
DOI: 10.6688/JISE.202007_36(4).0014

Processing Location-Based Aggregate Queries in Road Networks

YUAN-KO HUANG
Department of Maritime Information and Technology
National Kaohsiung University of Science and Technology
Kaohsiung, 811 Taiwan
E-mail: huangyk@nkust.edu.tw

In recent years, the research community has introduced various methods for processing the location-based queries on a single type of objects in road networks. However, in real-life applications user may be interested in obtaining information about different types of objects, in terms of their neighboring relationship. The sets of different types of objects closer to each other are termed the heterogeneous neighboring object sets (or *HNO sets* for short). To provide users with object information by considering both the spatial closeness of objects to the query object and the neighboring relationship between objects, we present useful and important *location-based aggregate queries* on finding the *HNO sets* in road net-works. The location-based aggregate queries are the *shortest average distance query* (*SAvgDQ*), the *shortest minimal distance query* (*SMinDQ*), the *shortest maximal distance query* (*SMaxDQ*), and the *shortest sum distance query* (*SSumDQ*). We first utilize a grid index to manage information of data objects and road networks, and then propose the *SAvgDQ*, *SMinDQ*, *SMaxDQ*, and *SSumDQ* processing algorithms, which are combined with the grid index to efficiently process the four types of location-based aggregate queries, respectively. A comprehensive set of experiments is conducted to demonstrate the efficiency of the proposed processing algorithms using real road network datasets.

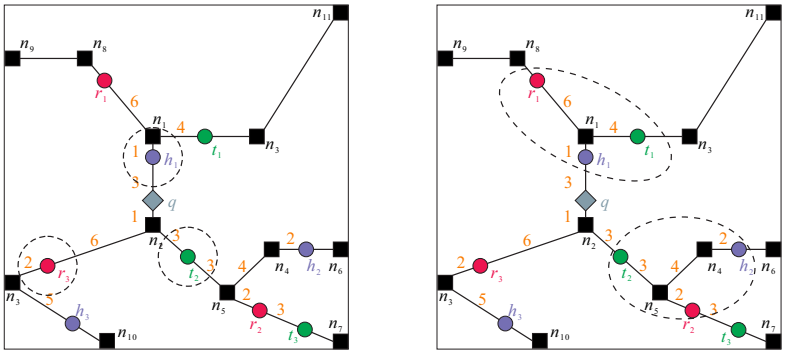
Keywords: location-based queries, road networks, heterogeneous neighboring object sets, location-based aggregate queries, grid index

1. INTRODUCTION

Due to the increasing need for real-time information of spatial data sets in road networks, efficient processing of various types of *location-based queries* has become imperative and essential in recent years [1-3] . Currently, most of the location-based queries in road networks take into account only a single type of spatial objects. In other words, the different types of objects (termed the *heterogeneous objects*) are independently considered in processing the location-based queries. Take a road network consisting of a set of edges and a set of nodes shown in Fig. 1 as an example. There are three types of objects in this road network, hotels, restaurants, and theaters. Assume that the user q wants to stay in a hotel, have lunch in a restaurant, and go to the movies. Based on the nearest neighbor of q for each object type, the objects h_1, r_3, and t_2 are retrieved as the result set because of their shortest road distance to q, as shown in Fig. 1 (a). However, in terms of the neighboring relationship between the chosen object set $\{h_1, r_3, t_2\}$, the road distance among themselves is greater than that of the other two object sets $\{h_1, r_1, t_1\}$

Received August 13, 2018; revised November 4, 2018, May 12, 2019; accepted May 27, 2019.
Communicated by Jianliang Xu.

and $\{h_2, r_2, t_2\}$, shown in Fig. 1(b). As the user would like to visit hotel, restaurant, and theater, the object sets $\{h_1, r_1, t_1\}$ and $\{h_2, r_2, t_2\}$ are actually better than the object set $\{h_1, r_3, t_2\}$. Unfortunately, the conventional location-based queries in road networks cannot provide such object information as all of them consider only the spatial closeness of the heterogeneous objects to the query object but ignore the neighboring relationship between the heterogeneous objects.



(a) NN result set (b) Better result set

Fig. 1. Example of location-based queries in road networks.

In order to provide useful information of spatial objects in road networks, we present important location-based queries, namely the *location-based aggregate queries*, in which both the neighboring relationship between the heterogeneous objects and the spatial closeness of the heterogeneous objects to the query object play important roles in determining the query result. For preserving the neighboring relationship between the heterogeneous objects, the location-based aggregate queries aim at finding the heterogeneous objects closer to each other by constraining their road distance to be within a user-defined distance d. Here, we term a set of objects satisfying the constraint of distance d the *heterogeneous neighboring object set* (or *HNO set*).

On the other hand, for maintaining the spatial closeness of the heterogeneous objects to the query object, four types of location-based aggregate queries are presented to provide information of *HNO set* according to specific user requirement. They are the *shortest average distance query* (*SAvgDQ*), the *shortest minimal distance query* (*SMinDQ*), the *shortest maximal distance query* (*SMaxDQ*), and the *shortest sum distance query* (*SSumDQ*). Consider the n types of objects, O_1, O_2, ..., O_n. Assume that there are m *HNO sets*, $\{o_1^1, o_2^1, ..., o_n^1\}$, $\{o_1^2, o_2^2, ..., o_n^2\}$, ..., $\{o_1^m, o_2^m, ..., o_n^m\}$, where $o_i^j \in O_i$, $i = 1 \sim n$, and $j = 1 \sim m$. Given a query object q, a set of objects $\{o_1^j, o_2^j, ..., o_n^j\}$ among these m *HNO sets* is determined, such that (1) for the *SAvgDQ*, the average road distance of $\{o_1^j, o_2^j, ..., o_n^j\}$ to q is shortest, (2) for the *SMinDQ* (*SMaxDQ*), the road distance of an object $o_i^j \in \{o_1^j, o_2^j, ..., o_n^j\}$ to q is shortest, where o_i^j is the nearest (furthest) neighbor of q, and (3) for the *SSumDQ*, the traveling distance, starting from q to visit each object in $\{o_1^j, o_2^j, ..., o_n^j\}$ exactly once, is the shortest.

To efficiently process the location-based aggregate queries, it is necessary to design effective strategies that can find the *HNO set* with the shortest average, minimal, maximal, or sum distance to the query object, without the need to (1) consider all object sets in determining the *HNO sets* and (2) compute the road distances between q and all objects contained in the *HNO sets*. To achieve this goal, we first design a grid index for the

road network, which is built by balancing the storage responsibility among grid cells, and several data structures to adequately maintain information of data objects and road network in each grid cell. Then, we develop a road network traversal in a best-first manner combined with the grid index to efficiently find the *HNO set* with the shortest average, minimal, maximal, or sum distance, by accessing only a small proportion of data objects in the road network.

This paper represents a major extension of our previous work [4], which focuses only on processing the shortest average distance query ($i.e.$, *SAvgDQ*) in road networks. The research reported in this paper could be viewed as a step toward supporting more general location-based queries in road networks. To sum up, the major contributions of this paper are as follows.

- Four types of location-based aggregate queries, the *SAvgDQ*, *SMinDQ*, *SMaxDQ*, and *SSumDQ* are presented for finding the *HNO set* with the shortest road distance in road networks. These queries can provide useful object information by considering both the spatial closeness of objects to the query object and the neighboring relationship between objects.

- We design appropriate data structures to represent a road network consisting of edges and nodes, and to store information of data objects in road network. Moreover, a grid index is used for accessing the data structures, so as to facilitate the traversal of the road network.

- Four efficient algorithms are developed, all of which operate based on a road network traversal in a best-first manner to retrieve the *HNO set* with the shortest average, minimal, maximal, and sum distance to the query object.

- We evaluate the query performance of the proposed algorithms with extensive experimental studies, by using real road network datasets.

2. RELATED WORK

Most of the conventional location-based queries concentrate on discovering the spatial closeness of objects to the query object. The range query is a well-known query, which can be used to find a set of objects that are inside a spatial region specified by the user. Recently, many efforts have been made on processing the range queries in different research domains, such as mobile information systems and uncertain database systems. The *K*NN query is the most common type of location-based queries, as it has important applications to the provision of location-based services. Many variations of *K*NN query have been proposed to provide the *K*-nearest neighbor information in numerous applications [5-7]. Some recent works consider both the spatial closeness of objects to the query object and the neighboring relationship between objects. Zhang $et\ al.$ [8] present the *KNG* query to determine the query result based on the minimum distance between data objects and the query object (referred to as inter-group distance) and the maximum distance among the data objects (referred to as inner-group distance). To appropriately keep the spatial closeness and the neighboring relationship of objects, in our previous work [9], the location-based aggregate queries are presented to obtain information of objects. However, the previous work focuses on the Euclidean spaces rather than the *road networks*.

In recent years, processing the location-based queries (such as the range and the KNN queries) in road networks has received considerable attention [3, 10, 11]. Nevertheless, the above works take into account only the spatial closeness of objects to the query object. The collective spatial keyword query [12, 13] is presented to retrieve a set of objects that collectively cover user-given keywords with the minimum cost. Moreover, Su $et\ al.$ [14] propose the group-based collective keyword (GBCK) query in road networks, considering not only the spatial closeness of objects to the query object but also the neighboring relationship between objects. The object group retrieved by the GBCK query is likely to be close to the query object but far away from each other, or close to each other but far away from the query object. As the GBCK query is inherently different from the location-based aggregate queries, it cannot be applied to find the group of objects that we address in this paper.

3. INDEX STRUCTURES

The grid index provides a foundation for our processing algorithms, which is also used in [4]. To make this paper self-contained, we briefly discuss the used grid index and data structures. The reader may refer to [4] for details. The n types of data objects are located in a road network, which is represented as an undirected weighted graph consisting of a set of nodes and a set of edges. A grid index is designed to efficiently manage information of objects, nodes, and edges, by partitioning the road network into multiple gird cells, each of which stores data of objects, nodes, and edges intersecting it. In order to balance the storage load of each grid cell, the data space is divided into a set of grid cells C by considering two parameters, P_{edge} and P_{obj}. For each grid cell c_i , the number of edges intersecting c_i cannot be greater than the parameter P_{edge}, and the number of objects enclosed in c_i must be less than the parameter P_{obj}. By exploiting the two parameters, the storage overhead for maintaining information of objects and road network can be evenly distributed among the grid cells.

The data space covering the road network is first divided into a set of grid cells C, and then for each grid cell $c_i \in C$, information of the network and data objects is stored in two tables, the edge table T_{edge} and the object table T_{obj}. T_{edge} stores the intersection information of each edge e with the grid cell c_i, including (1) the nodes n_i and n_j connecting e, where n_i is enclosed in c_i, (2) the length len of e, (3) the grid cell c_j enclosing the node n_j, and (4) a set S_c of grid cells intersecting e, except for the two grid cells c_i and c_j. Note that T_{edge} needs not be maintained if the grid cell c_i does not contain any node ($i.e.$, it contains the objects only). T_{obj} maintains the information of each object o enclosed in the grid cell c_i, including (1) the edge e containing o, (2) the node n connecting e and closed in c_i (if not, n is the node closer to o than another connecting node), and (3) the distance $dist$ between object o and node n.

The *HNO sets* have to be determined on-the-fly according to the user-defined distance d, which imposes more burdens on processing the location-based aggregate queries in road networks. To support the task of determining the *HNO sets* on-the-fly, two tables are used to maintain the distance information of objects and grid cells in the road network. The first table, denoted as T^d_{obj}, is used to store the road distance between any two objects, computed by using the Dijkstra's algorithm or the A* algorithm. The second table, denoted as T^d_{cell}, is designed to maintain the minimal distance and the maximal distance between two sets of objects, O_{c_i} and O_{c_j}, which are contained in two cells c_i and c_j, respectively.

4. PROCESSING ALGORITHMS

In this section, we first present the general procedure of the processing algorithms for the *SAvgDQ*, *SMinDQ*, *SMaxDQ*, and *SSumDQ* in detail. Then, we give a running example of the processing algorithm for the *SAvgDQ*, as the other three processing algorithms have the similar process to generate the query result.

4.1 General Procedure of Processing Algorithms

In all the four processing algorithms, a road network traversal in a best-first manner is developed to perform a network expansion starting from the edge containing the query object q and examine whether the *HNO set* with the shortest distance to q can be found from the objects visited so far. To achieve early termination in query processing, a min-heap H is used to keep each entry with the following information: (1) the edge identifier e, (2) the nodes n_i and n_j connecting the edge e (3) the grid cells c_i and c_j enclosing the nodes n_i and n_j respectively, and (4) the road distances $d(q, n_i)$ and $d(q, n_j)$ of n_i and n_j to q, respectively. The entries in H are sorted in ascending order of the road distance $d(q, n_i)$ and initially the information about the edge containing q is inserted into H. In each iteration, the first entry of H in the form of $(e, n_i, n_j, c_i, c_j, d(q, n_i), d(q, n_j))$ is de-heaped, and then the following three steps are sequentially executed to find the query result. **Step 1:** computing the road distances between the objects on e and the query object q. **Step 2:** determining whether the *HNO sets* exist among the objects visited so far. **Step 3:** checking whether the object set with the shortest distance can be found from the *HNO sets* discovered so far.

During the course of Step 1, the n types of objects that have been considered, with their road distance to the query object q, are separately kept in the lists, L_1, L_2, ..., L_n. The objects in each list are sorted according to their road distance $d(q, o)$ to q. When an entry of H in the form of $(e, n_i, n_j, c_i, c_j, d(q, n_i), d(q, n_j))$ is de-heaped from H, information of each object o on the edge e can be retrieved by accessing the grid index. For each object o on edge e, its distance $d(q, o)$ to q can be computed based on the following equation. Then, the entry $(o, d(q, o))$ is inserted into o's corresponding list among the lists L_1 to L_n. and the procedure proceeds to the next step.

$$d(q,o) = \begin{cases} d(q,n_i) + o.dist & \text{if } o \text{ is enclosed in } c_i, \\ d(q,n_j) - o.dist & \text{if } o \text{ is enclosed in } c_j, \\ d(q,n_i) + o.dist & \text{if } o \text{ is enclosed in } c' \in S_c \\ & \text{and } n_i \text{ is closer to } o \text{ than } n_j, \\ d(q,n_j) - o.dist & \text{otherwise.} \end{cases}$$

The goal of Step 2 is to determine whether the *HNO sets* exist among the objects visited so far. We utilize two data structures related to the grid cells visited so far to facilitate the task of determining the *HNO sets*. The first is a list, denoted as L_{cell}, which is used to keep the visited grid cells in which at least one object can be found. The second is a table, denoted as T_c, for each visited grid cell c in L_{cell}, which is used to store information in the form of $((o, o'), c')$, where (o, o') refers to an object pair indicating that the road distance between o and o' is less than or equal to the distance d, and o and o' are enclosed in c and c', respectively. Consider a de-heaped entry $(e, n_i, n_j, c_i, c_j, d(q, n_i), d(q, n_j))$ of H. Once the grid cell c_j encloses at least one object and is not in L_{cell}, it is inserted into L_{cell}. Also, each grid cell appearing in S_c of edge e is inserted into L_{cell} if it has not yet been visited. Then, the grid cells in L_{cell} are checked for the satisfaction of the

distance constraint d, so as to update their table T_c. Here, we use the two tables T^d_{obj} and T^d_{cell} mentioned in the previous section to improve the performance of updating the table T_c.

Suppose that the tables T_c and $T_{c'}$ of grid cells c and c' in L_{cell} need to be updated. Based on the maximal distance $d_M(c_{i,j})$ and the minimal distance $d_m(c_{i,j})$ between c and c' kept in T^d_{cell}, there are three cases to be considered: (1) $d_M(c_{i,j}) \leq d$, (2) $d_m(c_{i,j}) > d$, and (3) $d_m(c_{i,j}) \leq d < d_M(c_{i,j})$. For Case 1, each object pair $((o, o'), c')$ (or $((o', o), c)$) can be directly added into T_c (or $T_{c'}$), without the need to compare their distance with d. For Case 2, each object pair (o, o') cannot satisfy the requirement of *HNO set*, and thus the tables T_c and $T_{c'}$ need not be updated. For Case 3, the distance $d(o, o')$ needs to be derived from the table T^d_{obj}, and only the object pairs whose $d(o, o') \leq d$ are inserted into T_c and $T_{c'}$. Having updated the table T_c for each grid cell in L_{cell}, if there exist the *HNO sets* among the objects visited so far, then the average, minimal, maximal, and sum distance of each *HNO set* to the query object q can be estimated. The *HNO set* with the shortest distance to q is treated as the candidate and the procedure proceeds to the next step. If no *HNO set* exists, each of the unvisited edges connecting the node n_j is inserted into H and the procedure goes back to Step 1 for considering the next entry of H.

Step 3 aims to verify whether the candidate set discovered so far is the actual result for the *SAvgDQ*, *SMinDQ*, *SMaxDQ*, and *SSumDQ*. As the four types of location-based aggregate queries focus on different distance considerations, Step 3 needs to be designed accordingly so as to achieve early termination in query processing.

For the *SAvgDQ*: let $d_{avg}(q, \{o_1, o_2, ..., o_n\})$ be the average road distance of the candidate set $\{o_1, o_2, ..., o_n\}$ to the query object q, which can be estimated as

$$d_{avg}(q, \{o_1, o_2, ..., o_n\}) = \frac{1}{n}(\sum_{i=1}^{n} d(q, o_i)),$$

where $d(q, o_i)$ refers to the road distance between q and o_i. Also, let $(o^1_\alpha, d(q, o^1_\alpha))$ be the first entry of the α-th list L_α (meaning that o^1_α has the minimal road distance to q among the visited objects belonging to the α-th type of objects). Consider an unvisited object o_α belonging to the α-th type of objects. It has no chance to be included in the *SAvgDQ* result if its road distance $d(q, o_\alpha)$ to q satisfies the following equation:

$$d(q, o_\alpha) > n \times d_{avg}(q, \{o_1, o_2, ..., o_n\}) - \sum_{1 \leq \beta \leq n \wedge \alpha \neq \beta} d(q, o^1_\beta).$$

As o_α is an unvisited object, its road distance $d(q, o_\alpha)$ must be greater than the distance $d(q, n_j)$ of the node n_j to q (*i.e.*, $d(q, o_\alpha) > d(q, n_j)$). Motivated by this, we can infer that the unvisited edges connecting the node n_j need not be inserted into the min-heap H once the following equation holds:

$$d(q, n_j) > n \times d_{avg}(q, \{o_1, o_2, ..., o_n\}) - \min\{\sum_{1 \leq \beta \leq n \wedge \alpha \neq \beta} d(q, o^1_\beta) | \alpha = 1 \sim n\}.$$

Otherwise, the unvisited edges connecting the node n_j would be inserted into H for consideration.

For the *SMinDQ*: for an object set $\{o_1, o_2, ..., o_n\}$ that has still not been considered, it can be the *HNO set* if the distance between any two objects in $\{o_1, o_2, ..., o_n\}$ does not

exceed d. The above condition implies that for each object $o_\alpha \in \{o_1, o_2, ..., o_n\}$, its road distance $d(q, o_\alpha)$ to q must satisfy the following equation:

$$d(q, o_{NN}) \leq d(q, o_\alpha) \leq d(q, o_{NN}) + d,$$

where $d(q, o_{NN})$ refers to the road distance between q and its nearest neighbor $o_{NN} \in \{o_1, o_2, ..., o_n\}$, which is represented as

$$d(q, o_{NN}) = \min\{d(q, o_i) | i = 1 \sim n\}.$$

Compared to a candidate *HNO set* $\{o'_1, o'_2, ..., o'_n\}$ discovered so far, the set $\{o_1, o_2, ..., o_n\}$ will become the new candidate only when its minimal distance $d(q, o_{NN})$ is less than that (denoted as $d(q, o'_{NN})$) of $\{o'_1, o'_2, ..., o'_n\}$. Conversely, $\{o_1, o_2, ..., o_n\}$ cannot be the *SMinDQ* result because $d(q, o_{NN}) > d(q, o'_{NN})$. As a result, an unvisited object o_α cannot appear in the *SMinDQ* result once its road distance $d(q, o_\alpha)$ satisfies the following equation:

$$d(q, o_\alpha) > d(q, o'_{NN}) + d.$$

This is because for each *HNO set* containing o_α, the difference between its minimal distance $d(q, o_{NN})$ and $d(q, o_\alpha)$ does not exceed d, and thus $d(q, o_{NN})$ must be greater than $d(q, o'_{NN})$ if the above equation holds. For the sake of early termination of the road network traversal, the unvisited edges connecting the node n_j need not be inserted into the min-heap H as the following equation holds:

$$d(q, n_j) > d(q, o'_{NN}) + d.$$

Otherwise, the unvisited edges connecting n_j have to be inserted into H.

For the *SMaxDQ*: the result would be affected by the road distance between q and its *furthest* neighbor in each *HNO set*. Let object o_{FN} be the furthest neighbor of q in a set of object $\{o_1, o_2, ..., o_n\}$ and its road distance to q is equal to $d(q, o_{FN})$, which is computed as

$$d(q, o_{FN}) = \max\{d(q, o_i) | i = 1 \sim n\}.$$

If object o_{FN} is farther to q than the furthest neighbor o'_{FN} in the candidate set $\{o'_1, o'_2, ..., o'_n\}$ ($i.e.$, $d(q, o_{FN}) > d(q, o'_{FN})$), then the set $\{o_1, o_2, ..., o_n\}$ cannot be the *SMaxDQ* result no matter whether it is a *HNO set* or not. As such, an unvisited object o_α needs not be considered if the following equation holds:

$$d(q, o_\alpha) > d(q, o'_{FN}),$$

which also implies that the unvisited edges connecting the node n_j is not inserted into the min-heap H when

$$d(q, n_j) > d(q, o'_{FN}).$$

For the *SSumDQ*: suppose that among the *HNO sets* discovered so far, the set of objects $\{o'_1, o'_2, ..., o'_n\}$ has the minimal traveling distance from q, which is defined and computed as follows:

$$d(q, \{o'_1, o'_2, ..., o'_n\}) = \min\{d(q, o'_i) + d(o'_i, HNO) | i = 1 \sim n\},$$

where $d(o'_i, HNO)$ refers to the shortest traveling distance starting from o'_i to visit each object in $\{o'_1, o'_2, ..., o'_n\} \setminus \{o'_i\}$ exactly once, which can be obtained by looking up the table T^d_{obj}. For a set of objects $\{o_1, o_2, ..., o_n\}$ containing an unvisited object o_α, it cannot be the *SSumDQ* result because the road distance of o_α to q is greater than the shortest traveling distance from q to the candidate set $\{o'_1, o'_2, ..., o'_n\}$. That is,

$$d(q, o_\alpha) > d(q, \{o'_1, o'_2, ..., o'_n\}),$$

which means that the minimal road distance to reach o_α from q (passing the other objects exactly once) must also exceed $d(q, \{o'_1, o'_2, ..., o'_n\})$. Similarly, the unvisited edges connecting the node n_j needs not be inserted into the min-heap H once

$$d(q, n_j) > d(q, \{o'_1, o'_2, ..., o'_n\}).$$

The three steps mentioned above ($i.e.$, Step 1, Step 2, and Step3) are sequentially executed on each de-heaped entry until H is empty. If there exists at least one *HNO set*, then the candidate set is returned as the query result. Otherwise, no object set can satisfy the requirement of *HNO set*. The pseudo code of detailed steps for the processing algorithms is given in Algorithm 1.

Algorithm 1: Processing algorithm

```
Input  : A grid index, a distance d, and the query object q
Output: A HNO set with the shortest distance to q

initialize a heap H;
insert information of the edge containing q into H;
while ( H is not empty ) do
    de-heap (e, n_i, n_j, c_i, c_j, d(q, n_i), d(q, n_j)) from H;
    /* executing Step 1                                          */
    foreach object o on e do
        compute d(q, o);
        insert (o, d(q, o)) into the corresponding list among L_1 to L_n;
    /* executing Step 2                                          */
    insert c_j and cells in S_c of e into L_cell;
    foreach cell pair (c, c') in L_cell do
        check for the constraint of d to update T_c and T_c';
    if there exist HNO sets then
        foreach each HNO set {o_1, o_2, ..., o_n} do
            compute the average, minimal, maximal, and sum distance of
            {o_1, o_2, ..., o_n} to q;
        set {o_1, o_2, ..., o_n} with the shortest distance as a candidate;
    /* executing Step 3                                          */
    if the inequality for d(q, n_j) does not hold then
        insert information of the edges connecting n_j into H;
if there exists a candidate {o_1, o_2, ..., o_n} then
    return {o_1, o_2, ..., o_n};
```

4.2 Running Example

We use a concrete example to illustrate how the *SAvgDQ* result can be efficiently determined, where the query object q is located at the node n_2 enclosed in the grid cell

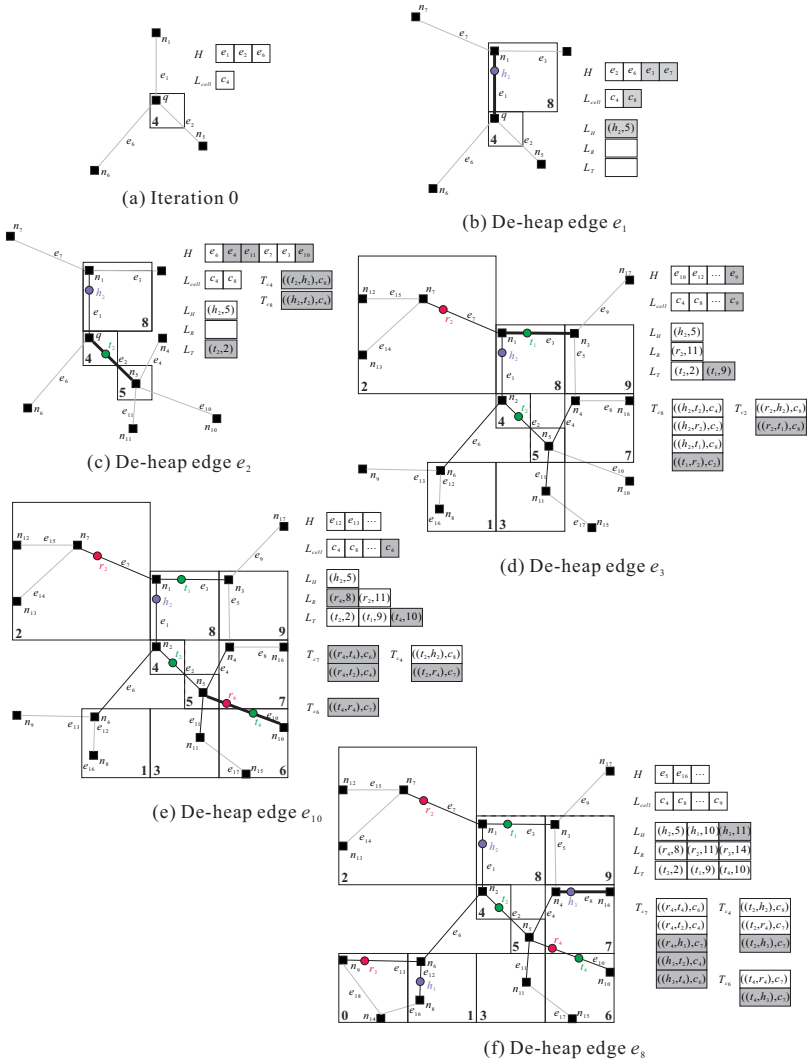
c_4, and the distance d is set to 11. Initially, the min-heap H contains information of the three edges e_1, e_2, and e_6 connecting the query object q, as shown in Fig. 2(a). In the first iteration, the first entry $(e_1, q, n_1, c_4, c_8, 0, 6)$ of H is de-heaped, on which Step 1, Step 2, and Step 3 are sequentially executed (refer to Fig. 2(b)). In Step 1, the road distance $d(q, h_2)$ between objects h_2 and q is computed as 5, and then $(h_2, 5)$ is inserted into the list L_H. Having executed Step 2, the gird cells c_4 and c_8 are kept in the list L_{cell}. Then, the edges e_3 and e_7 connecting the node n_1 are inserted into H and Step 3 is skipped as no *HNO set* has been discovered so far. In the second iteration (refer to Fig. 2(c)), where the edge e_2 is considered, the object t_2 with $d(q, t_2)$ is added into the corresponding list L_T (after Step 1). By looking up the table T^d_{cell}, we know that $d_M(c_{4,8}) < d$ for grid cells c_4 and c_8, and thus the entries $((t_2, h_2), c_8)$ and $((h_2, t_2), c_4)$ are added into the tables T_{c_4} and T_{c_8}, respectively (after Step 2). The procedure proceeds to de-heap the first entry of H, which is omitted here due to space limitations. In the sequel, when the edge e_3 is de-heaped from H (as shown in Fig. 2(d)), the entry $(t_1, 9)$, meaning that the road distance $d(q, t_1)$ is equal to 9, is inserted into L_T. After Step 2, we can find a *HNO set*, $\{h_2, r_2, t_1\}$, whose average road distance to q is equal to $\frac{25}{3}$. Therefore, Step 3 is executed to determine whether $\{h_2, r_2, t_1\}$ is the actual *SAvgDQ* result. Unfortunately, the road distance $d(q, n_3)$ does not satisfy the termination condition, so that the edges e_5 and e_9 have to be inserted into H for further consideration. As shown in Fig. 2(e), after checking the edge e_{10}, the objects r_4 and t_4 are added into the lists L_R and L_T, respectively, and also the table T_c for each grid cell in L_{cell} is updated accordingly. When the edge e_8 is de-heaped from H (refer to Fig. 2(f)), the object h_3 on e_8 leads to two new *HNO sets*, $\{h_3, r_4, t_2\}$ and $\{h_3, r_4, t_4\}$, and the former has a smaller average distance. In Step 3, as the road distance $d(q, n_{16})$ satisfies the termination condition, the edges connecting n_{16} need not be en-heaped in H. Finally, the algorithm terminates once H is empty and $\{h_3, r_4, t_2\}$ is returned as the *SAvgDQ* result.

5. PERFORMANCE EVALUATION

In this section, four sets of experiments are conducted to demonstrate the efficiency of the proposed methods. We investigate the effects of four important factors on the query performance. They are the *values of P_{edge} and P_{obj}*, the *number of objects*, the *number of data types* ($i.e.$, n), and the *value of distance* d.

5.1 Performance Settings

All the experiments are performed on a PC with Intel 2.80 GHz and 4GB RAM. The algorithms are implemented in C++. Two road networks are used in our simulation. The first is the Oldenburg (a city in Germany), consisting of about 6000 nodes and 7000 edges. The second is the San Joaquin County with about 18200 nodes and 23800 edges. The data set, generated using the generator proposed in [15], consists of n types of objects (where n varies from 1 to 5), and the number of objects for each type ranges from 1K to 30K. The data space is divided into multiple grid cells by considering the parameters P_{edge} and P_{obj}, in which the pair (P_{edge}, P_{obj}) changes from $(25, 25)$ to $(1000, 1000)$. In the experimental space, we also generate 30 query objects on the edges of road network and each of them issues a *SAvgDQ*, a *SMinDQ*, a *SMaxDQ*, or a *SSumDQ* to the server, where the distance d varies from 0.1% to 3% of the entire space. Then, we present the LBAQ (stands for location-based aggregate queries) algorithm to process the 30 queries. The performance is measured by the average CPU time and the average ratio of accessed grid cells ($i.e.$, $\frac{N_{cell}}{N_{total}}$, where N_{cell} refers to the number of accessed grid cells and N_{total}



(a) Iteration 0

(b) De-heap edge e_1

(c) De-heap edge e_2

(d) De-heap edge e_3

(e) De-heap edge e_{10}

(f) De-heap edge e_8

Fig. 2. Running example of processing the *SAvgDQ*.

is the total number of grid cells) in performing workloads of the 30 queries. In our simulation, we compare the LBAQ algorithm, with a naive algorithm which is composed of the following steps: (1) all of the *HNO sets* in the road network are determined by exploiting the distance information kept in the table T^d_{obj}, (2) for each *HNO set*, its average, minimal, maximal, and sum distance to the query object is computed using the A* algorithm, and (3) the *HNO set* with the shortest distance is returned as the query result. Table 1 summarizes the parameters under investigation, along with their default values and ranges.

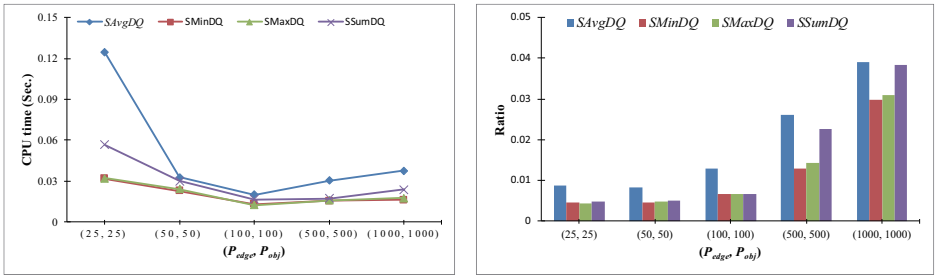
5.2 Effect of Parameters P_{edge} and P_{obj}

Fig. 3 studies the impact of P_{edge} and P_{obj} (affecting the number of grid cells), on the performance of processing the four types of location-based aggregate queries. In this set of experiments, we vary the pair (P_{edge}, P_{obj}) from $(25, 25)$ to $(1000, 1000)$ and measure the CPU cost and the ratio of accessed grid cells for the proposed processing algorithms.

Table 1. System parameters.

Parameter	Default	Range
(P_{edge}, P_{obj})	$(100, 100)$	$(25, 25)$, $(50, 50)$, $(100, 100)$, $(500, 500)$, $(1000, 1000)$
Number of objects (K)	10	1, 5, 10, 20, 30
n	3	1, 2, 3, 4, 5
d (%)	1	0.1, 0.5, 1, 2, 3

Fig. 3(a) illustrates the CPU times for processing the *SAvgDQ*, *SMinDQ*, *SMaxDQ*, and *SSumDQ*, respectively. The CPU time first decreases and then slightly increases with the increasing P_{edge} and P_{obj}. This is mainly because (1) the smaller P_{edge} and P_{obj} result in a larger number of grid cells, so that more combinations of grid cells to be considered in determining the *HNO sets*, while (2) the larger P_{edge} and P_{obj} lead to more information about the edges and objects for each grid cell, and thus more CPU time is required for computing the road distance of objects to the query object. On the other hand, the increasing P_{edge} and P_{obj} make the ratio $\frac{N_{cell}}{N_{total}}$ increase, which is illustrated in Fig. 3(b). Actually, both N_{cell} and N_{total} decrease, but the value of N_{cell} decreases slower than that of N_{total}. That's why $\frac{N_{cell}}{N_{total}}$ exhibits an increasing trend. To decide an appropriate pair (P_{edge}, P_{obj}) to build the grid index for query processing, we observe that when $(P_{edge}, P_{obj}) = (100, 100)$, the improvement of CPU time becomes insignificant, while the ratio of accessed grid cells increases noticeably. As the experimental result reveals that $(100, 100)$ is a better choice than the others, we will use it as the default (P_{edge}, P_{obj}) in all the rest experiments.



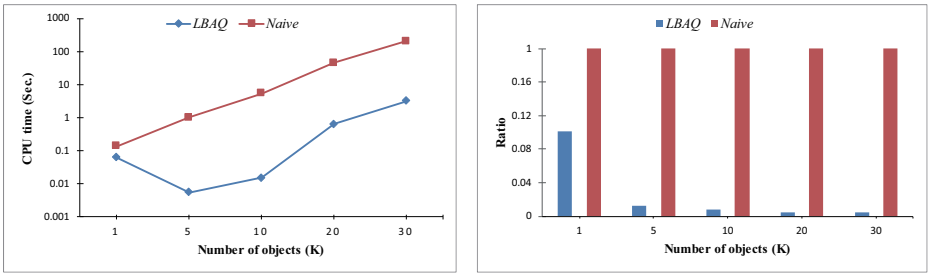
(a) CPU time (b) Ratio

Fig. 3. Effect of parameters P_{edge} and P_{obj}.

5.3 Effect of Number of Objects

The second set of experiments investigates the effect of the number of objects for each data type on the performance of the LBAQ algorithm and the naive algorithm. Fig. 4(a) measure the CPU time for the LBAQ algorithm compared to the naive algorithm, by varying the object number from 1K to 30K. In all experiments, for the naive algorithm the CPU cost increases significantly with the number of objects, because when the object number increases, more distance computations are required for finding the *HNO set* with the shortest distance. An interesting observation from the experimental result is that for the LBAQ algorithm the CPU cost first decreases and then increases, exhibiting a V-shape. This is mainly because (1) for a smaller number of objects, there are fewer *HNO*

sets in the road network, which leads to more objects to be visited in query processing, and (2) for a larger number of objects, more *HNO sets* have the similar distance, and thus results in more CPU time spent on determining the query result. As shown in Fig. 4(b), the ratio $\frac{N_{cell}}{N_{total}}$ for the naive algorithm remains constant ($i.e.$, $\frac{N_{cell}}{N_{total}} = 1$) regardless of the number of objects. This is because no matter what the object quantity is, all grid cells storing object information have to be accessed for determining the query result. As for the LBAQ algorithm, the ratio $\frac{N_{cell}}{N_{total}}$ shows a decreasing trend, because a larger number of objects (leading to more *HNO sets*) increases the chance that the *HNO set* with the shortest distance can be found by accessing only the grid cells closer to the query object. The experimental results also show that the proposed algorithm have a significantly better performance on both the CPU cost and the ratio $\frac{N_{cell}}{N_{total}}$ compared to the naive algorithm.



(a) CPU time (b) Ratio

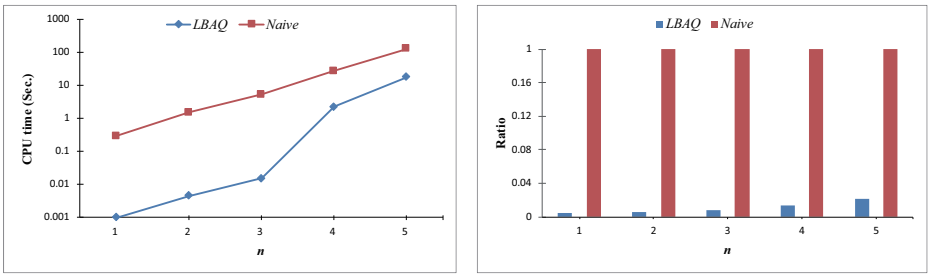
Fig. 4. Effect of number of objects.

5.4 Effect of Number of Data Types

The third set of experiments shown in Fig. 5 illustrates the performance of the LBAQ algorithm and the naive algorithm as a function of the number of data sources (ranging n from 1 to 5). When $n = 1$ ($i.e.$, only a single data type), the task of determining whether the road distance between objects is greater than d is no longer needed. It implies that the problems of processing the *SAvgDQ*, *SMinDQ*, *SMaxDQ*, and *SSumDQ* can all be reduced to finding the nearest neighbor of the query object. The better performance of the LBAQ algorithm, compared to the naive algorithm, demonstrates that the LBAQ algorithm can be successfully applied to the environment in which user is interested in obtaining information about a single data type. When the number of data sources n gets larger than 1, both the CPU time and the ratio $\frac{N_{cell}}{N_{total}}$ for the LBAQ algorithm increase, because the value of n dominates the total number of combinations of object sets. Nevertheless, the experimental results show that the performance of the LBAQ algorithm still outperforms its competitor under various numbers of n, which confirms again that the road network traversal we develop in this paper can efficiently improve the query performance by considering only a small proportion of the *HNO sets*.

5.5 Effect of Distance d

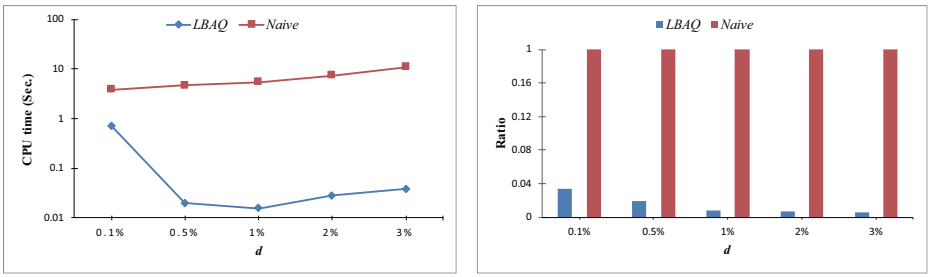
The set of experiments studies the CPU time and the ratio $\frac{N_{cell}}{N_{total}}$ under different values of the distance d. In the experiments, we vary the value of d from 0.1% to 3% of the experimental space. As shown in Fig. 6, the curve for the LBAQ algorithm has the V-shape trends in terms of the CPU time (while the curve for the naive algorithm shows an increasing trend) and the decreasing trend in terms of the ratio $\frac{N_{cell}}{N_{total}}$. For a smaller value



(a) CPU time (b) Ratio

Fig. 5. Effect of number of data types.

of d, most of the object sets cannot satisfy the requirement of *HNO set* and thus (1) for the naive algorithm, less number of *HNO sets* needs to be considered, while (2) for the LBAQ algorithm, it has to access more object sets and involve more road distance computations of object sets. Nevertheless, the LBAQ algorithm still outperforms the naive algorithm by a significant amount, which can be attributed to the early termination in query processing. For a larger value of d, the differences in the CPU time between the LBAQ algorithm and the naive algorithm become noticeable, as shown in Fig. 6(a). This is because for a larger d each object set has a higher chance to be the *HNO set* so that for the LBAQ algorithm the query result could be determined earlier by only considering the object sets closer to the query object. That's also why the LBAQ algorithm yield a better performance than the naive algorithm, in terms of the ratio $\frac{N_{cell}}{N_{total}}$ (shown in Fig. 6(b)).

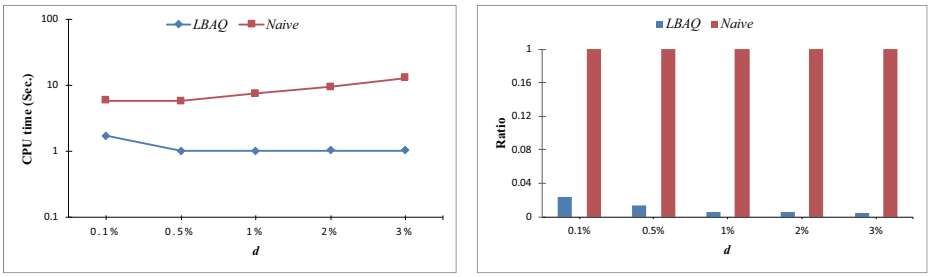


(a) CPU time (b) Ratio

Fig. 6. Effect of distance d.

5.6 Performance for Larger Road Network

We conduct a set of experiments to study how well the LBAQ algorithm and the naive algorithm work for a larger road network, the San Joaquin County. Fig. 7(a) and Fig. 7(b) measure the CPU time and the ratio $\frac{N_{cell}}{N_{total}}$, as a function of d (varying from 0.1% to 3% of the experimental space). Compared to the experimental result in Fig. 6 (illustrating the performance for a small road network), the CPU time for the both the algorithms increases, while the ratio $\frac{N_{cell}}{N_{total}}$ decreases. The reason for the higher CPU overhead is that the road connectivity of the larger network is more complicated so that more time is spent on computing the road distance. On the other hand, the decrease of the ratio $\frac{N_{cell}}{N_{total}}$ for the larger network is because the queries can be terminated by accessing fewer cells closer to the query object.



(a) CPU time (b) Ratio

Fig. 7. Performance for larger road network.

6. CONCLUSIONS

This paper focused on processing the four types of location-based aggregate queries on the *HNO sets* in road networks, where the road distance between objects is computed based on the connectivity of the network. We designed a grid index to manage information of data objects located in a road network consisting of edges and nodes, and made use of the grid index to access only a small proportion of data objects while computing the result of location-based aggregate queries. The processing algorithms, associated with the grid index, were developed to retrieve the *HNO set* with the shortest distance to the query object. Comprehensive experiments demonstrated the efficiency of the proposed processing algorithms.

ACKNOWLEDGMENT

This work was supported by Ministry of Science and Technology, Taiwan under Grants MOST 107-2119-M-992-304 and MOST 108-2621-M-992-002.

REFERENCES

1. X. Fu, X. Miao, J. Xu, and Y. Gao, "Continuous range-based skyline queries in road networks," *World Wide Web*, Vol. 20, 2017.
2. M. Kolahdouzan and C. Shahabi, "Voronoi-based k nearest neighbor search for spatial network databases," in *Proceedings of the 13th international conference on Very Large Data Bases*, 2004, pp. 840-851.
3. D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *Proceedings of the 29th International Conference on Very Large Data Bases*, 2003, pp. 802-813.
4. Y.-K. Huang, C.-H. Su, C. Lee, and C.-H. Ho, "Efficient evaluation of shortest average distance query on heterogeneous neighboring objects in road networks," in *Proceedings of the 21st ACM International Database Engineering and Applications Symposium*, 2017, pp. 209-218.
5. R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis, "Nearest neighbor and reverse nearest neighbor queries for moving objects," *VLDB Journal*, Vol. 15, 2006, pp. 229-249.

6. Y. Chen and J. M. Patel, "Efficient evaluation of all-nearest-neighbor queries," in *Proceedings of IEEE 23rd International Conference on Data Engineering*, 2007, pp. 1056-1065.
7. D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *Proceedings of the 20th International Conference on Data Engineering*, 2004, pp. 301-312.
8. D. Zhang, C.-Y. Chan, and K.-L. Tan, "Nearest group queries," in *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, 2013.
9. Y.-K. Huang, "Location-based aggregate queries for heterogeneous neighboring objects," *IEEE Access*, Vol. 5, 2017, pp. 4887-4899.
10. Y.-K. Huang and L.-F. Lin, "Efficient processing of continuous min-max distance bounded query with updates in road networks," *Information Sciences*, Vol. 278, 2014, pp. 187-205.
11. H. Hu, D. L. Lee, and J. Xu, "Fast nearest neighbor search on road networks," in *Extending Database Technology*, 2006, pp. 186-203.
12. Y. Gao, J. Zhao, B. Zheng, and G. Chen, "Efficient collective spatial keyword query processing on road networks," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 17, 2016, pp. 469-479.
13. C. Long, R. C.-W. Wong, K. Wand, and A. W.-C. Fu, "Collective spatial keyword queries: A distance owner-driven approach," in *International conference on ACM SIGMOD*, 2013.
14. S. Su, S. Zhao, X. Cheng, R. Bi, X. Cao, and J. Wang, "Group-based collective keyword querying in road networks," *Information Processing Letters*, Vol. 118, 2017, pp. 83-90.
15. T. Brinkhoff, "A framework for generating network-based moving objects," *GeoInformatica*, Vol. 6, 2002, pp. 153-180.

Yuan-Ko Huang received the BS and Ph.D. degrees from National Cheng-Kung University, Taiwan, in 2004 and 2009, respectively. He joined the faculty of National Kaohsiung University of Science and Technology in 2016 and is currently a Professor of the Department of Maritime Information and Technology. His research interests are in the areas of mobile data management, spatio-temporal databases, and big data analysis. He has published many papers in major database journals and conferences, such as Information Systems, GeoInformatica, Information Sciences, SSDBM, IDEAS, and so on.