JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 37, 975-990 (2021)
DOI: 10.6688/JISE.202107_37(4).0014

Efficient Evaluation of Minimum Total Cost Queries on Heterogeneous Neighboring Objects

YUAN-KO HUANG
Department of Maritime Information and Technology
National Kaohsiung University of Science and Technology
Kaohsiung, 824 Taiwan
E-mail: huangyk@nkust.edu.tw

In recent years, many of the *location-based services* provide information of a single type of spatial objects, based on their spatial closeness to the query object. However, in real-life applications, user may be interested in obtaining information about different types of objects ($e.g.$, hotels, restaurants, and theaters), in terms of their neighboring relationship. Moreover, an important aspect that has not been previously explored is the total cost of experiencing various types of spatial objects. As a result, we present a new type of location-based queries, named the *minimum total cost query* (*MTCQ*), which takes both the neighboring relationship and the total experiencing cost of spatial objects into account. Given the n types of spatial objects and a user-defined distance d, the *MTCQ* finds a set of n objects such that the distance between any pair of objects does not exceed d and their total cost is smallest. To efficiently process the *MTCQ*, we utilize an R-tree-based index, the R^c-$tree$, to manage the spatial objects with their locations and costs. Then, two processing algorithms, namely the *top-k-based MTCQ algorithm* and the *enhanced MTCQ algorithm*, are proposed to determine a set of objects satisfying the constraint of distance d, whose total cost is lowest. Finally, extensive experiments using the synthetic dataset are conducted to demonstrate the efficiency and the effectiveness of the proposed algorithms.

Keywords: location-based services, minimum total cost query, R^c-$tree$, top-k-based MTCQ algorithm, enhanced MTCQ algorithm

1. INTRODUCTION

In recent years, the *location-based services* focus on efficiently managing a large number of spatial objects, and then providing various types of location-based queries [1, 2, 3, 4]. For example, the range queries and the nearest neighbor queries can be used to find the objects within a query range and the closest object to the query object, respectively. There are many applications related to the location-based services, such as location-aware advertisements, traffic control systems, and geographical information systems. Most of the processing techniques for the location-based queries consider a single type of objects ($e.g.$, hotels, restaurants, or theaters). However, some users may not be interested in obtaining information of one type of objects. Instead, they want to know information about different types of objects. As a result, in [5], we consider

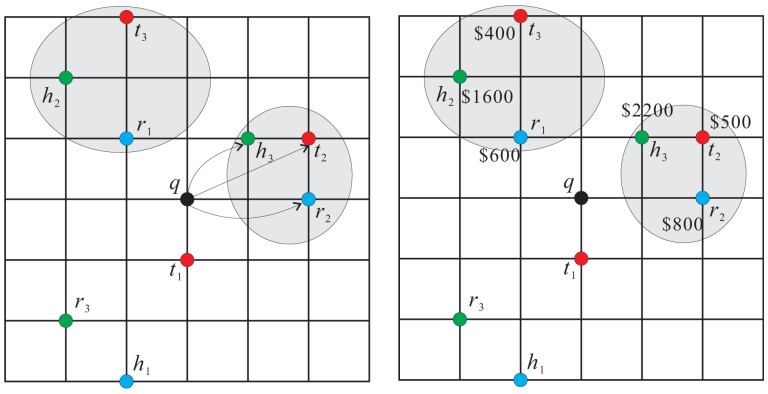
Received August 2, 2019; revised November 28, 2019 & April 7, 2020; accepted April 27, 2020.
Communicated by Hung-Yu Kao.

the different types of objects, termed the *heterogeneous neighboring objects* (*HNOs* for short), and present the *location-based aggregate queries* on the *HNOs*. The *HNOs* and the location-based aggregate queries are defined as follows.

- Consider the n types of spatial objects, O_1, O_2, ..., O_n. If there is a set of objects, $\{o_1, o_2, ..., o_n\}$, where o_i belongs to O_i and $i = 1 \sim n$, and the distance between any pair of objects in this set is less than or equal to a user-defined distance d, then the objects set $\{o_1, o_2, ..., o_n\}$ is a set of *HNOs*.

- Assume that there are m sets of *HNOs*. Given a query object q, the location-based aggregate queries retrieve a set of objects $\{o_1, o_2, ..., o_n\}$, among the m sets of *HNOs*, such that the *average*, *min*, *max*, or *sum* distance of $\{o_1, o_2, ..., o_n\}$ to q is minimal.

Fig. 1(a) shows an example of processing the location-based aggregate queries on the *HNOs*, where the use-defined distance is set to 2 and the set of *HNOs* with the shortest average distance to q will be retrieved. There are three types of objects in the space, hotels h_1 to h_3, restaurants r_1 to r_3, and theatres t_1 to t_3. As $d = 2$, only the two object sets $\{h_2, r_1, t_3\}$ and $\{h_3, r_2, t_2\}$ can be the sets of *HNOs*. By comparing their average distances to the query object q, the set $\{h_3, r_2, t_2\}$ is returned as the query result because it has the shorter distance. From the above figure, we know that the result of the location-based aggregate queries is mainly based on the distance of the *HNOs* ($i.e.$, hotels, restaurants, and theatres) to the query object. However, in many real applications, the users may want to experience the facilities while keeping the total cost as low as possible. In such applications, a set of *HNOs* with the minimum experiencing cost in total is the best choice. Let us consider the example in Fig. 4(b). As we can see, although the set $\{h_3, r_2, t_2\}$ is closer to q than the set $\{h_2, r_1, t_3\}$, its total cost ($i.e.$, $2200 + 800 + 500 = 3500$) is much higher than that of $\{h_2, r_1, t_3\}$ ($i.e.$, $1600 + 600 + 400 = 2600$). As such, the set $\{h_2, r_1, t_3\}$ is the better choice if the cost is a main concern in determining the query result.



(a) Location-based aggregate queries. (b) Min cost queries.

Fig. 1. Different types of location-based queries.

In this paper, we present a new type of location-based queries on the *HNOs*, named the *minimum total cost query* (*MTCQ* for short), to find the set of *HNOs* having the smallest total cost. Formally, the *MTCQ* is defined as follows.

- Consider the n types of spatial objects, O_1, O_2, ..., O_n, where O_i has a *cost* attribute. Based on the user-defined distance d, there are m sets of *HNOs*. The *MTCQ* retrieves a set of *HNOs* $\{o_1, o_2, ..., o_n\}$, among the m sets of *HNOs*, such that the total cost of $\{o_1, o_2, ..., o_n\}$ is smallest.

To efficiently process the *MTCQ*, we first utilize an R-tree-based index, the R^c-$tree$, to manage the spatial objects with their locations and costs. Then, two processing algorithms, namely the *top-k-based MTCQ algorithm* and the *enhanced MTCQ algorithm*, are proposed to determine a set of objects satisfying the constraint of distance d ($i.e.$, a set of *HNOs*), whose total cost is lowest. The *top-k-based MTCQ algorithm* is designed based on traversing the R^c-$tree$ for each type of spatial objects to retrieve the top-k objects with the smallest cost. Having constructed the k^n sets of n objects, the set of *HNOs* with the lowest total cost is returned as the query result. Thus, the performance of the *top-k-based MTCQ algorithm* would be affected by the number of object types ($i.e.$, the value of n). To effectively alleviate the above effect, the *enhanced MTCQ algorithm* is proposed by taking advantage of a simultaneous traversal of the R^c-$trees$ built on the n types of spatial objects. Moreover, three criteria, the *d-pruning criterion*, the *d-qualifying criterion*, and the *cost-pruning criterion*, are devised to improve the query performance of the *enhanced MTCQ algorithm*. To sum up, the major contributions of this paper are as follows.

- A new type of location-based queries, the *MTCQ*, is presented to provide information of spatial objects with the better neighboring relationship and the lowest cost in total.
- The R^c-$tree$ is used to manage each type of spatial objects, which is built by taking into account the spatial objects' locations and costs.
- The *top-k-based MTCQ algorithm* and the *enhanced MTCQ algorithm* are proposed to efficiently process the *MTCQ* by traversing the R^c-$trees$.
- Extensive experiments using the synthetic dataset are conducted to demonstrate the efficiency and the effectiveness of the proposed algorithms.

The remainder of this paper is organized as follows. In Sections 2, we review related work on processing the location-based queries. In Section 3, we describe the data structures of the R^c-$tree$ and the pruning criteria used for the processing algorithms. Then, the *top-k-based MTCQ algorithm* and the *enhanced MTCQ algorithm* are presented in Section 4. Section 5 shows extensive experiments on the performance of the proposed approaches. Finally, we conclude this paper in Section 6.

2. RELATED WORK

In this section, we first survey the related works for processing the location-based queries, in which the spatial closeness of the spatial objects to the query object plays an

important role in query processing. Then, we discuss the processing techniques for the location-based queries, where the neighboring relationship of spatial objects is a main concern in determining the query result. Finally, we review some works on processing the collective spatial keyword queries.

The *K-nearest neighbor query* (*K*NN query) [6, 7] is the most popular type of location-based queries, which is presented to retrieve the *K* spatial objects with the best spatial closeness to the query object (that is, the *K* spatial objects that are closest to the query object). Recently, several variations of *K*NN query have been proposed to provide information of *K*-nearest neighbors in numerous applications. To address the issue of scalability, the *all-nearest-neighbors query* (ANN query) [8] focuses on finding the *K*-nearest neighbors for all objects in a query set. The ANN query inevitably incurs more CPU and I/O overhead because multiple *K*NN queries are executed. To express requests by groups of users, the *aggregate nearest neighbor query* has been proposed in [9], which is defined as follows. Given a set of query objects Q and a set of objects O, the aggregate nearest neighbor query retrieves the spatial object in O, so that an aggregate distance function ($e.g.$, sum, min, or max) with respect to all objects in Q is minimized. The *nearest surrounder query* [10] finds the nearest neighbor around a query object, from the perspective of the query object's orientation. In other words, the nearest surrounder query retrieves the nearest neighbors of a query object at different angles. The *range nearest-neighbor query* [11] is related to but different from the *K*NN query, where a query object is replaced by a query region. Specifically, the range nearest-neighbor query finds the nearest neighbors for every point in a spatial region, instead of a point. Another variation of *K*NN query with asymmetric property is the *reverse nearest neighbor query* (RNN query) [1, 12], where the set of objects whose nearest neighbor is the query object would be returned as the query result. The *within query* is proposed in [13, 14] to maintain the better spatial closeness of spatial objects by constraining their distance to the query object to be within a user-defined distance d.

Then, we discuss some of the location-based queries that aim at preserving the good neighboring relationship between spatial objects. Given two data sources S_1 and S_2, the *K closest pair query* [15] focuses on finding the K closest object pairs between S_1 and S_2 ($i.e.$, the K pairs (a, b), where $a \in S_1$ and $b \in S_2$, with the smallest distance between them). The *spatial join query* [16] determines a set of object pairs that satisfy a user-defined spatial predicate ($e.g.$, *overlap* or *coverage*). In [17], the spatial join query is extended to process the *multiway spatial join query*, in which the spatial predicate is a function over m data sources (where $m \geq 2$). Recently, the *k nearest group query* in [18] tries to preserve the spatial closeness of objects to the query object and the neighboring relationship between objects, by computing the sum of the *minimum* distance between objects and the query object and the *maximum* distance among the objects. However, the *k* nearest group query may return a set of objects that are close to the query object but far away from each other, or are close to each other but far away from the query object. As a result, the location-based aggregate queries in [5] are further presented to appropriately keep the spatial closeness and the neighboring relationship of spatial objects.

Recently, the collective spatial keyword query [19, 20] is presented to find a set of objects that collectively cover user-given keywords with the minimum cost. Moreover, Su $et\ al.$ [21] propose the group-based collective keyword (GBCK) query, 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 could 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 *MTCQ*, it cannot be applied to find the group of objects that we address in this paper.

3. INDEX STRUCTURE AND PRUNING CRITERIA

In this section, we first describe the data structures of the R^c-$tree$, which is used as the underlying index structure for the *top-k-based MTCQ algorithm* and the *enhanced MTCQ algorithm*. Then, we discuss the three pruning criteria, the *d-pruning criterion*, the *d-qualifying criterion*, and the *cost-pruning criterion*, which are devised to upgrade the query performance of the *enhanced MTCQ algorithm*.

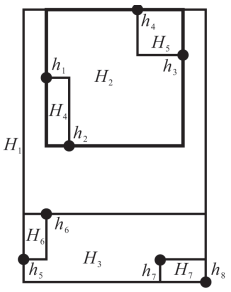
3.1 Data Structures of R^c-$tree$

The R^c-$tree$ is a height-balanced index structure, where objects are recursively grouped in a bottom-up manner according to objects' locations and costs. Each entry of a leaf node of a R^c-$tree$ has the structure $((o.x, o.y)$, $o.c$, $o.ptr)$, where $(o.x, o.y)$ refers to the location of spatial object o, $o.c$ is the cost of object o, and $o.ptr$ is a pointer to the actual object tuple in the database. Each entry of an internal node of the R^c-$tree$ has the structure $(MBR_E, E.c_m, E.ptr)$, where MBR_E is a minimum bounding rectangle (represented as (x_l, y_d, x_r, y_u)) enclosing all the objects in the child node E of this internal node, $E.c_m$ is the minimum among all costs of the objects enclosed in MBR_E, and $E.ptr$ is a pointer to node E.

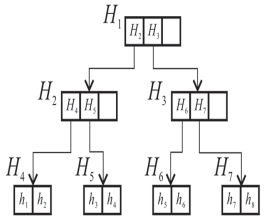
Let us use the example in Fig. 2 to illustrate the information maintained for the R^c-$tree$. As shown in Fig. 2(a), eight hotels h_1 to h_8 in the space are indexed by the R^c-$tree$. Initially, hotels h_1 to h_8 are grouped according to their locations and costs into four leaf nodes H_4 to H_7. Take the leaf node H_4 as an example. As hotels h_1 and h_2 are enclosed by MBR_{H_4}, they are the entries of the leaf node H_4 and will be stored as $((7, 15), 1800)$ and $((8, 12), 2000)$, respectively. Then, the leaf nodes H_4 to H_7 are recursively grouped into two internal nodes, H_2 and H_3, that becomes the entries of the root. Because the extent of MBR_{H_2} covers hotels h_1 to h_4, the node H_2 is maintained in the form of $(MBR_{H_2}, 1500)$, where 1500 represents the minimal cost among the four hotels. The corresponding structure of the R^c-$tree$ (for the hotels) is shown in Fig. 2(b), and the complete information of the leaf and internal nodes is illustrated in Fig. 2(c).

3.2 Three Pruning Criteria

Consider the n types of spatial objects, O_1, O_2, ..., O_n, which are separately indexed by the n R^c-$trees$, termed *the* R^c_1-$tree$, *the* R^c_2-$tree$, ..., and *the* R^c_n-$tree$. Let $\{s_1, s_2, ..., s_n\}$ be a set of entries to be considered, where entry s_i corresponds to a MBR E_i or an object o_i indexed by the R^c_i-$tree$. The goal of the first criterion, the ***d-pruning criterion***, is to prune the set $\{s_1, s_2, ..., s_n\}$ that cannot satisfy the constraint of distance d, without the need to compute all pairwise distances between entries. Two parameters, d_x and d_y, are used in the *d-pruning criterion*. The parameter d_x refers to the minimal distance between the two entries that are furthest apart on the x-dimension. As for d_y, it is the minimal distance between the two entries that are furthest apart on the y-dimension.



(a) Hotels in space.



(b) R^c-$tree$ for hotels.

Internal node

id	MBR_E	$E.c_m$
H_1	MBR_{H1}	1500
H_2	MBR_{H2}	1500
H_3	MBR_{H3}	1400
H_4	MBR_{H4}	1800
H_5	MBR_{H5}	1500
H_6	MBR_{H6}	1400
H_7	MBR_{H7}	1600

Leaf node

id	$(o.x, o.y)$	$o.c$
h_1	(7, 15)	1800
h_2	(8, 12)	2000
h_3	(13, 16)	1600
h_4	(11, 18)	1500
h_5	(6, 7)	1800
h_6	(7, 9)	1400
h_7	(12, 7)	1600
h_8	(14, 6)	1800

(c) Information of nodes.

Fig. 2. Data structures of the R^c-$tree$.

Assume that $R^{\vdash}$ and $L^{\dashv}$ are the left boundary of the *rightmost* entry and the right boundary of the *leftmost* entry, respectively. Then, the distance d_x is computed as:

$$d_x = \begin{cases} R^{\vdash} - L^{\dashv} & \text{if } R^{\vdash} > L^{\dashv}, \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, let $U^{\perp}$ and $D^{\top}$ be the lower boundary of the *uppermost* entry and the upper boundary of the *lowermost* entry, respectively. Then, the distance d_y can be obtained using the following equation:

$$d_y = \begin{cases} U^{\perp} - D^{\top} & \text{if } U^{\perp} > D^{\top}, \\ 0 & \text{otherwise.} \end{cases}$$

With the two distances d_x and d_y, the set of entries $\{s_1, s_2, ..., s_n\}$ can be pruned using the *d-pruning criterion* if $\max(d_x, d_y) > d$.

The second criterion, the ***d-qualifying criterion***, is used to efficiently determine a set of entries $\{s_1, s_2, ..., s_n\}$ satisfying the constraint of distance d, without computing the distance between all pairs of entries. To achieve this, a rectangle R, whose lower-left corner and upper-right corner are (x_l, y_d) and (x_r, y_u) respectively, is constructed to tightly enclose the extents of all entries. With the rectangle R, the distance between any two objects enclosed in R does not exceed the distance between the two points $(R.x_l, R.y_d)$ and $(R.x_r, R.y_u)$. As such, the set of entries $\{s_1, s_2, ..., s_n\}$ must satisfy the constraint of distance d if the following equation holds: $(R.x_r - R.x_l)^2 + (R.y_u - R.y_d)^2 \leq d^2$.

The third criterion is the ***cost-pruning criterion***, which is designed to prune the non-qualifying object set, no matter whether it is a set of *HNOs* or not. Let $\{o'_1, o'_2, ..., o'_n\}$ be a set of *HNOs* found so far, whose total cost, defined as $TC(o'_1, o'_2, ..., o'_n)$, is equal to $\sum_{i=1}^{n} o'_i.c$. Consider a set of MBRs $\{E_1, E_2, ..., E_n\}$. As mentioned in Section 3.1, $E_i.c_m$ is the minimum among all costs of the objects enclosed in the MBR E_i. Thus, the cost of an object enclosed in E_i must be greater than or equal to $E_i.c_m$. It means that if the sum of $E_i.c_m$ for $1 \leq i \leq n$, defined as $TC_m(E_1, E_2, ..., E_i)$ (*i.e.*, $\sum_{i=1}^{n} E_i.c_m$),

is greater than the total cost $TC(o'_1, o'_2, ..., o'_n)$, then all the object sets consisting of the objects enclosed in MBRs E_1, E_2, ..., E_n cannot be the query result. Motivated by this, a set of entries $\{s_1, s_2, ..., s_n\}$ can be pruned using the *cost-pruning criterion* when (1) $TC_m(E_1, E_2, ..., E_i) > TC(o'_1, o'_2, ..., o'_n)$ for s_i corresponds to a MBR E_i or (2) $TC(o_1, o_2, ..., o_i) > TC(o'_1, o'_2, ..., o'_n)$ for s_i corresponds to an object o_i, without the need to check for the constraint of distance d.

Fig. 3 shows how to prune the non-qualifying object sets using the three pruning criteria. As shown in Fig. 3(a), the *d-pruning criterion* is imposed on a set of MBRs $\{H, R, T\}$ and a set of objects $\{h, r, t\}$, respectively. As the MBRs H and R are the *rightmost* and *leftmost entries*, respectively, the distance d_x is equal to the minimal horizontal distance between H and R. Similarly, the distance d_y is represented as the minimal vertical distance between H and R because they are also the *lowermost* and *uppermost* entries, respectively. When the set of objects $\{h, r, t\}$ is considered, the distance d_x (d_y) is computed as the minimal horizontal (vertical) distance between objects h and t (h and r). If one of the two distances d_x and d_y of the set $\{H, R, T\}$ (or $\{h, r, t\}$) exceeds the distance d, then it is pruned by the *d-pruning criterion*. Consider the example in Fig. 3(b), in which the *d-qualifying criterion* is used to check whether the sets of MBRs $\{H, R, T\}$ and objects $\{h, r, t\}$ satisfy the constraint of distance d. A rectangle whose length of diagonal is equal to $\sqrt{(x_r - x_l)^2 + (y_u - u_d)^2}$ is constructed to enclosed the set $\{H, R, T\}$ (or $\{h, r, t\}$). Once the diagonal length is less than or equal to d, $\{H, R, T\}$ (or $\{h, r, t\}$) is guaranteed to satisfy the constraint of distance d. Fig. 3(c) shows that $\{h', r', t'\}$ is a set of *HNOs* found so far, whose total cost $TC(h', r', t') = 2400$. According to the *cost-pruning criterion*, the set of MBRs $\{H, R, T\}$ can be pruned by the *HNOs* set $\{h', r', t'\}$ as its $TC_m(H, R, T)$ is greater than $TC(h', r', t')$. Also, a set of objects $\{h, r, t\}$ is pruned in the same way because of $TC(h, r, t) > TC(h', r', t')$.

4. PROCESSING ALGORITHMS

Given the n types of spatial objects, O_1, O_2, ..., O_n, where O_i has a cost attribute, and the user-defined distance d, the *MTCQ* is used to find a set of *HNOs* $\{o_1, o_2, ..., o_n\}$, such that the total cost of $\{o_1, o_2, ..., o_n\}$ is smallest. To efficiently process the *MTCQ*, we propose two processing algorithms, the *top-k-based MTCQ algorithm* and the *enhanced MTCQ algorithm*, which are described separately as follows.

Algorithm 1: Top-k-based MTCQ algorithm

Input : The n types of objects indexed by the R^c-$trees$ and a distance d
Output: The *HNOs* set with the smallest total cost

foreach *type of objects* **do**
  traverse the R^c-$tree$ to find the top-k objects with the smallest cost;
construct the k^n sets of n objects;
foreach *set of n objects* **do**
  compute the distances between two objects to compare with the distance d;
return the *HNOs* set with the smallest total cost;

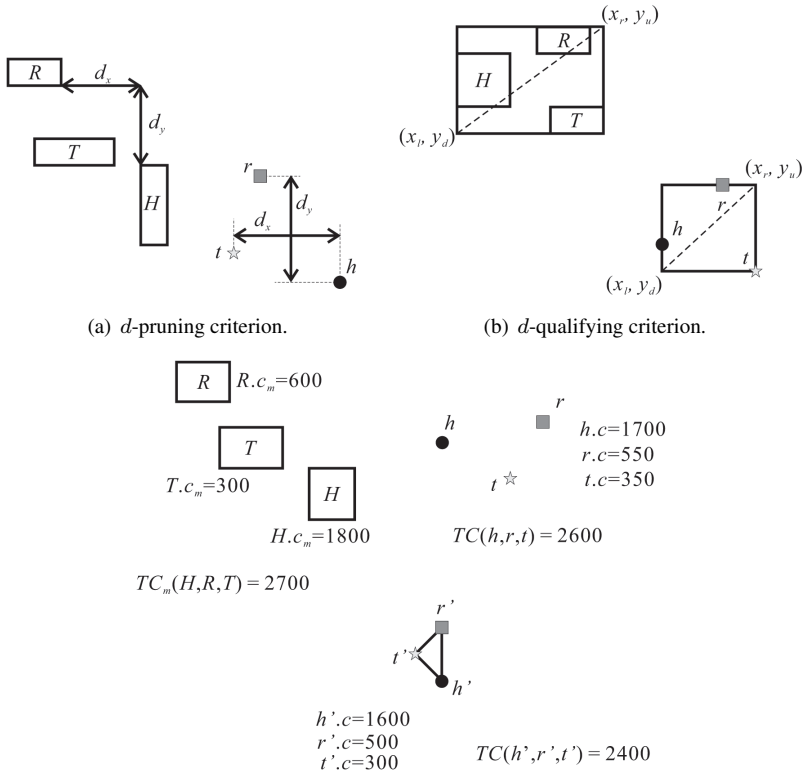


Fig. 3. Illustration of the pruning criteria.

4.1 Top-k-based MTCQ Algorithm

The *top-k-basedMTCQ algorithm* consists of the following three steps: (1) for each type of spatial objects, the top-k objects with the smallest cost are retrieved, so as to construct the k^n sets of n objects; (2) for each set of n objects, the distance between any two objects is computed and compared to the distance d; (3) the set of n objects satisfying the constraint of distance d and with the lowest total cost is returned as the *MTCQ* result.

To improve the performance of Step 1 ($i.e.$, finding the top-k objects for each object type), we perform a depth-first traversal of the R^c-$tree$ built on each type of objects to filter the non-qualifying objects. The procedure of Step 1 begins with the root node of the R^c-$tree$ and proceeds down the tree. When a MBR MBR_E is encountered, the following pruning criterion is used to determine whether MBR_E can be pruned or not. If $E.c_m$ of the MBR MBR_E is greater than the largest value in the costs of the top-k objects considered so far, then all the objects enclosed in MBR_E can be pruned, because their costs exceed that of the top-k objects. Consider again Fig. 3, where k is set to 2 ($i.e.$, finding the top-2 objects). Assume that hotels h_3 and h_4 are the top-2 objects considered so far. For the MBR MBR_{H_4}, as its $H_4.c_m$ is greater than $h_3.c$ and $h_4.c$ ($i.e.$,

$1800 > 1600$ and $1800 > 1500$), hotels h_1 and h_2 enclosed in MBR_{H_4} can be filtered using the designed pruning criterion.

For Step 2 ($i.e.$, checking whether a set of n objects satisfies the constraint of distance d), we design the following criterion to determine a set of n objects that must be a set of *HNOs*, without the need to exhaustively check all pairs of objects for whether their distances exceed the distance d. Let $\{o_1, o_2, ..., o_n\}$ be a set of n objects to be considered. Then, $\{o_1, o_2, ..., o_n\}$ *must* be a set of *HNOs* if

$$(x_r - x_l)^2 + (y_u - y_d)^2 \leq d^2,$$

where

$$x_r = \max\{o_i.x | i = 1 \sim n\}, x_l = \min\{o_i.x | i = 1 \sim n\},$$
$$y_u = \max\{o_i.y | i = 1 \sim n\}, y_d = \min\{o_i.y | i = 1 \sim n\}.$$

Although the *top-k-based MTCQ algorithm* can be used to retrieve the *MTCQ* result, it needs to traverse the R^c-$trees$ for the n types of spatial objects to construct the k^n sets of n objects. As a result, its performance must be dominated by the number of object types ($i.e.$, the value of n). To further improve the performance of processing the *MTCQ*, the *enhanced MTCQ algorithm* is proposed by taking advantage of a simultaneous traversal of the R^c-$trees$ built on the n types of spatial objects. The pseudo code for the *top-k-based MTCQ algorithm* is given in Algorithm 1.

Algorithm 2: Enhanced MTCQ algorithm

Input : The n types of objects indexed by the R^c-$trees$ and a distance d
Output: The *HNOs* set with the smallest total cost

```
initialize a sorted list L;
insert information of the root nodes of R^c-trees into L;
while ( L is not empty ) do
    retrieve ({s_1, s_2, ..., s_n}, flag, TC(s_1, s_2, ..., s_n)) from L;
    if each s_i corresponds to MBR E_i then
        decompose into the m sets of child entries;
        foreach set of child entries {s_1^j, s_2^j, ..., s_n^j} do
            impose the d-pruning criterion;
            impose the d-qualifying criterion to update flag;
            compute TC(s_1^j, s_2^j, ..., s_n^j);
    /* each s_i corresponds to object o_i                          */
    else
        if {o_1, o_2, ..., o_n} is a HNOs set then
            return {o_1, o_2, ..., o_n};
```

4.2 Enhanced MTCQ Algorithm

Recall that the n types of spatial objects, O_1, O_2, ..., O_n are separately indexed by the R_1^c-$tree$, the R_2^c-$tree$, ..., and the R_n^c-$tree$. The procedure of the *enhanced MTCQ*

algorithm begins with the root nodes of the R_1^c-$tree$, R_2^c-$tree$, ..., and R_n^c-$tree$ and proceeds down the trees simultaneously. During the traversal of the n R^c-$trees$, a sorted list L is used to maintain information of the sets of n entries considered so far. Note that each set of n entries consists of either n MBRs or n objects. Each element of L stores a set $\{s_1, s_2, ..., s_n\}$'s information, in the form of $(\{s_1, s_2, ..., s_n\}, flag, TC(s_1, s_2, ..., s_n))$, where $flag$ indicates whether $\{s_1, s_2, ..., s_n\}$ ***must*** satisfy the constraint of distance d, and $TC(s_1, s_2, ..., s_n)$ refers to (1) the minimal total cost $TC_m(E_1, E_2, ..., E_n)$ if each s_i corresponds to a MBR E_i or (2) the total cost $TC(o_1, o_2, ..., o_i)$ if each s_i corresponds to an object o_i. The elements of L are sorted in ascending order of their $TC(s_1, s_2, ..., s_n)$. Initially, L only contains information of the set $\{s_1, s_2, ..., s_n\}$, where s_i corresponds to the root node of the R_i^c-$tree$, and its $flag$ and $TC(s_1, s_2, ..., s_n)$ are both set to 0. In each iteration, the first element of L ($i.e.$, the set $\{s_1, s_2, ..., s_n\}$ whose $TC(s_1, s_2, ..., s_n)$ is the smallest among the sets in L) is retrieved. According to the entries comprising the set $\{s_1, s_2, ..., s_n\}$, there are two cases to be considered: (1) each entry s_i corresponds to MBR E_i and (2) each entry s_i corresponds to object o_i.

For the case that the set $\{s_1, s_2, ..., s_n\}$ consists of the n MBRs, each entry s_i is decomposed into its child entries, so as to construct the m sets of entries $\{s_1^1, s_2^1, ..., s_n^1\}$, $\{s_1^2, s_2^2, ..., s_n^2\}$, ..., $\{s_1^m, s_2^m, ..., s_n^m\}$. Then, the following three steps are processed sequentially. The first step is to impose the *d-pruning criterion* on the m sets of entries. Once a set $\{s_1^j, s_2^j, ..., s_n^j\}$ is pruned by the *d-pruning criterion*, it can be immediately discarded. Having checked the m sets of entries, the procedure proceeds to the next step, in which the remaining sets of entries are checked for the *d-qualifying criterion* to update their $flag$. If a set $\{s_1^j, s_2^j, ..., s_n^j\}$ passes the *d-qualifying criterion*, its $flag$ is set to 1 (meaning that it ***must*** satisfy the constraint of distance d). Otherwise, its $flag$ is equal to 0. Note that $flag$ of all the remaining sets can directly be set to 1 without running the *d-qualifying criterion* when their parent set $\{s_1, s_2, ..., s_n\}$'s $flag = 1$. The third step is to compute the value of $TC(s_1^j, s_2^j, ..., s_n^j)$ for each remaining set $\{s_1^j, s_2^j, ..., s_n^j\}$. Then, each remaining set is inserted into the list L with its $flag$ and $TC(s_1^j, s_2^j, ..., s_n^j)$.

For the case that each entry of the set $\{s_1, s_2, ..., s_n\}$ is an object ($i.e.$, an object set $\{o_1, o_2, ..., o_n\}$), the procedure consists of the following two steps. The first step is to determine whether $\{o_1, o_2, ..., o_n\}$ is a set of ***HNOs*** by looking up its $flag$. If its $flag$ is equal to 0, then the distance between any two objects is compared to the distance d. Once the distance between a pair of objects exceeds the distance d, $\{o_1, o_2, ..., o_n\}$ is discarded and the next iteration starts by retrieving the first element of L. Otherwise ($i.e.$, $flag = 1$), $\{o_1, o_2, ..., o_n\}$ must satisfy the constraint of distance d. Only if $\{o_1, o_2, ..., o_n\}$ is found as a set of ***HNOs***, the procedure proceeds to the next step. In the second step, all of the elements in L can be pruned by the set $\{o_1, o_2, ..., o_n\}$, based on the *cost-pruning criterion*. Finally, $\{o_1, o_2, ..., o_n\}$ is reported as the query result and the ***MTCQ*** is terminated. Algorithm 2 describes the pseudo code for the *enhanced MTCQ algorithm*.

5. PERFORMANCE EVALUATION

In this section, we conduct three sets of experiments to measure the efficiency of the *top-k-based MTCQ algorithm* and the *enhanced MTCQ algorithm*, by investigating the effects of three important factors on the performance of processing the ***MTCQ***. These

factors are *the number of spatial objects*, *the number of object types* ($i.e.$, the value of n), and *the value of distance d*. Moreover, we study the perfomance of the proposed methods using three real datasets. We first describe the performance settings and then show the experimental results with detailed discussions.

5.1 Performance Settings

All the experiments are performed on a PC with Intel 2.70 GHz CPU and 16GB RAM. The *top-k-based MTCQ algorithm* and the *enhanced MTCQ algorithm* are implemented in JAVA. A synthetic dataset and three real datasets are used in our simulation. The synthetic dataset has n types of spatial objects (where n varies from 1 to 5). Each type of spatial objects contains O (ranging from 1K to 300K) objects whose locations are spread over a region of $1,000,000 \times 1,000,000$ with uniform distribution. For the real datasets, the *Beijing*, *Manchester*, and *Pittsburgh* files (consisting of about 400K, 1000K, and 1200K objects, respectively) are obtained from the OpenStreetMap [22]. The cost of each object in synthetic and real datasets ranges between 100 and 2000. Based on the locations and costs of spatial objects, the R_1^c-$tree$, the R_2^c-$tree$, ..., and the R_n^c-$tree$, are built to index the n types of objects. In the experimental space, we randomly generate 30 query objects issuing the *MTCQ*, where the distance d changes from 0.01% to 5% of the entire space. The performance is measured by the average CPU time and the average number of node accesses of the R^c-$trees$ in performing workloads of the 30 queries. To compare the performance of the proposed algorithms, we present a baseline algorithm to process the *MTCQ*, in which the m sets of *HNOs* are first determined by computing the distances between any two objects to compare with the distance d, and then the *HNOs* set with the smallest total cost is returned as the query result. Table 1 summarizes the parameters under investigation, along with their default values and ranges.

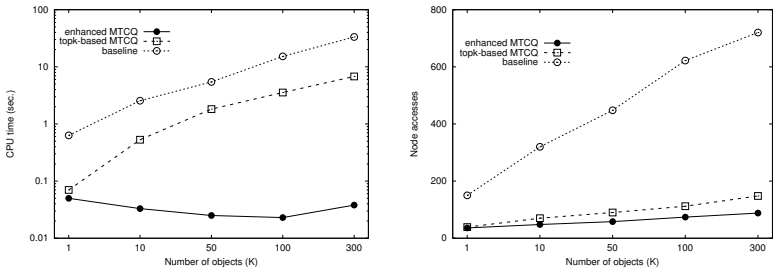
Table 1. System parameters.

Parameter	Default	Range
Number of objects (K)	100	1, 10, 50, 100, 300
Number of object types	3	1, 2, 3, 4, 5
Distance d (%)	0.1	0.01, 0.05, 0.1, 1, 5

5.2 Effect of Number of Objects

In this subsection, we measure the CPU time and the number of node accesses for the baseline algorithm, the *top-k-based MTCQ algorithm* and the *enhanced MTCQ algorithm* under various numbers of spatial objects (varying from 1K to 300K). As shown in Fig. 4(a), using a logarithmic scale for the y-axis, the curve for the *enhanced MTCQ algorithm* first decreases and then increases with the increase of object number. This is because for a smaller number of objects, fewer object sets can satisfy the constraint of distance d, and thus more distance computations are required for finding the sets of *HNOs*. For the baseline algorithm and the *top-k-based MTCQ algorithm*, the CPU time grows drastically because an increasing number of objects leads to more processing cost in finding the m sets of *HNOs* for the baseline algorithm and in determining the top-k objects

(note that it is executed n times) for the *top-k-based MTCQ algorithm*. Fig. 4(b) evaluates the node accesses for the three algorithms. All curves exhibit the increasing trends as a larger number of objects inevitably causes more index nodes to be accessed. Moreover, the experimental result shows a wide gap between the *enhanced MTCQ algorithm* and its competitors, confirming that using the one-time query evaluation efficiently improves the query performance.



(a) CPU time. (b) Number of node accesses.

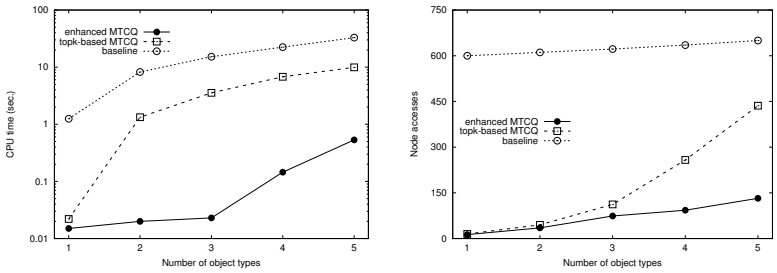
Fig. 4. Effect of number of spatial objects.

5.3 Effect of Number of Object Types

The set of experiments shown in Fig. 5 demonstrates the performance (including the CPU time and the number of node accesses) of the proposed algorithms as a function of the number of object types ($i.e.$, ranging n from 1 to 5). In the case that $n = 1$ (that is, considering only a single type of objects), the task of checking for the constraint of distance d is no longer needed. That is why the performance of the *top-k-based MTCQ algorithm* is almost as good as that of the *enhanced MTCQ algorithm* in terms of both the CPU time and the number of node accesses. On the other hand, when n is larger than 1 ($i.e.$, multiple object types are considered), the curves for the *top-k-based MTCQ algorithm* grow rapidly, compared to the *enhanced MTCQ algorithm*. The reason is that in the *top-k-based MTCQ algorithm* the repetitive executions are required and dominated by n, while the *enhanced MTCQ algorithm* is executed only once regardless of n. As for the baseline algorithm, it yields the worst performance in terms of the CPU time and the node access, because all of the object sets have to be accessed for determining the m sets of *HNOs*.

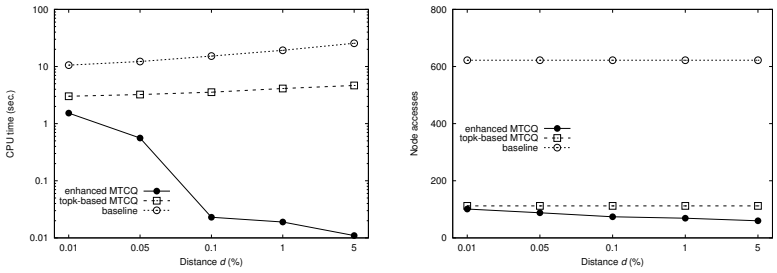
5.4 Effect of Distance d

As shown in Fig. 6, the set of experiments is conducted to study how the user-defined distance d affects the performance of processing the *MTCQ*, by varying the distance d from 0.01% to 5% of the experimental space. As we can see, the curves for the *enhanced MTCQ algorithm* show that a larger distance d results in a lower CPU time (in Fig. 6(a)) and a less number of node accesses (in Fig. 6(b)) when the *MTCQ* is processed. This improvement can be attributed to the fact that for a smaller d, most of the object sets cannot be the sets of *HNOs* so that the *enhanced MTCQ algorithm* needs to access more index nodes and involve more distance computations of non-qualifying object sets. Con-



(a) CPU time. (b) Number of node accesses.

Fig. 5. Effect of number of object types.



(a) CPU time. (b) Number of node accesses.

Fig. 6. Effect of distance d.

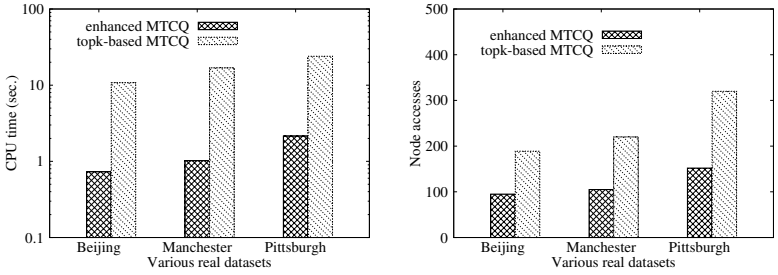
versely, a larger d increases the chance for each object set to be a set of *HNOs* and thus the *MTCQ* result could be determined early. The experimental results also illustrate that the performances of the baseline algorithm and the *top-k-based MTCQ algorithm* are quite insensitive to the distance d, but still fall behind the *enhanced MTCQ algorithm*.

5.5 Performance for Real Datasets

In this subsection, three sets of experiments are conducted to investigate the performance of the proposed methods using three real datasets, the *Beijing*, *Manchester*, and *Pittsburgh* datasets (containing about 400K, 1000K, and 1200K objects, respectively). As shown in Fig. 7, as the *Pittsburgh* dataset has a larger number of objects (compared to the *Beijing* and *Manchester* datasets), it demands more computation time and accesses more index node in processing the *MTCQ* for both the *top-k-based MTCQ algorithm* and the *enhanced MTCQ algorithm*. On the other hand, the *Beijing* dataset contains much fewer objects than the *Manchester* and *Pittsburgh* datasets, but incurs the slightly lower CPU cost and number of node accesses in comparison with the other two datasets. The reason is that the *Beijing* dataset has a denser object distribution so that it leads to more *HNOs* sets for the *enhanced MTCQ algorithm* and more processing time spent on finding the top-k objects for the *top-k-based MTCQ algorithm*.

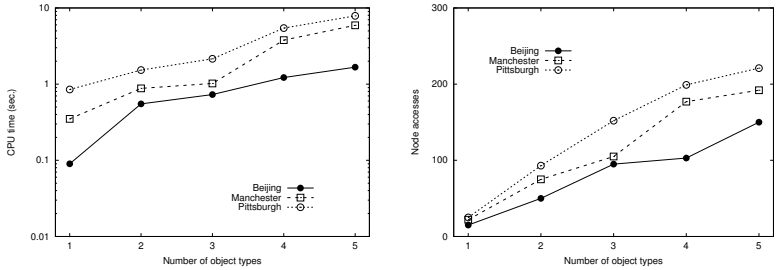
Figs. 8 and 9 study the effects of the number of object types and the distance d on the performance (in terms of CPU time and number of node accesses) of the *enhanced*

MTCQ algorithm, respectively, for the *Beijing*, *Manchester*, and *Pittsburgh* datasets. The experimental results show that for the *Pittsburgh* dataset, the *enhanced MTCQ algorithm* consumes higher CPU time (shown in Fig. 8(a) and Fig. 9(a)) and requires more node accesses (shown in Fig. 8(b) and Fig. 9(b)), compared to the *Beijing* and *Manchester* datasets.



(a) CPU time. (b) Number of node accesses.

Fig. 7. Effect of number of objects on real datasets.



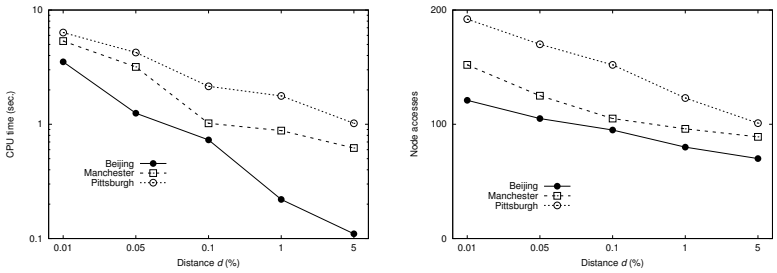
(a) CPU time. (b) Number of node accesses.

Fig. 8. Effect of number of object types on real datasets.

This is expected because the *Pittsburgh* dataset contains more objects, enforcing it demands more computation time and accesses more index node for determining the *HNOs* satisfying the constraint of distance d. As for the *Beijing* dataset, it has a better performance in all cases mostly because of its smaller cardinality. From the experimental results, we know that the *enhanced MTCQ algorithm* is also suitable for various real datasets.

6. CONCLUSIONS

In this paper, we focused on processing the *MTCQ* to find a set of *HNOs* with the lowest total cost. In order to efficiently process the *MTCQ*, the R^c-$tree$ was used as the underlying index for managing the locations and costs of spatial objects. Then, the *top-k-based MTCQ algorithm* and the *enhanced MTCQ algorithm*, combined with the R^c-$tree$, were proposed to retrieve the set of *HNOs* with the lowest cost in total. Comprehensive experiments demonstrated the efficiency of the proposed processing algorithms.



(a) CPU time. (b) Number of node accesses.

Fig. 9. Effect of distance d on real datasets.

ACKNOWLEDGMENT

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

REFERENCES

1. 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.
2. M. F. Mokbel, X. Xiong, and W. G. Aref, "Sina: Scalable incremental processing of continuous queries in spatio-temporal databases," in *Proceedings of International Conference on ACM SIGMOD*, 2004, pp. 623-634.
3. A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao, "Modeling and querying moving objects," in *Proceedings of IEEE International Conference on Data Engineering*, 1997, pp. 422-432.
4. Y. Tao and D. Papadias, "Time-parameterized queries in spatio-temporal databases," in *Proceeding of International Conference on ACM SIGMOD*, 2002, pp. 334-345.
5. Y.-K. Huang, "Location-based aggregate queries for heterogeneous neighboring objects," *IEEE Access*, Vol. 5, 2017, pp. 4887-4899.
6. G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *ACM Transactions on Database Systems*, Vol. 24, 1999, pp. 265-318.
7. N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *Proceedings of International Conference on ACM SIGMOD*, 1995, pp. 71-79.
8. Y. Chen and J. M. Patel, "Efficient evaluation of all-nearest-neighbor queries," in *Proceedings of IEEE International Conference on Data Engineering*, 2007, pp. 1056-1065.
9. D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *Proceedings of IEEE International Conference on Data Engineering*, 2004, pp. 301-312.
10. K. C. Lee, W.-C. Lee, and H. V. Leong, "Nearest surrounder queries," in *Proceedings of International Conference on Data Engineering*, 2006, pp. 1444-1458.

11. H. Hu and D. L. Lee, "Range nearest-neighbor query," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, 2006, pp. 78-91.
12. F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2000, pp. 201-212.
13. G. Iwerks, H. Samet, and K. Smith, "Continuous k-nearest neighbor queries for continuously moving points with updates," in *Proceedings of International Conference on Very Large Data Bases*, 2003, pp. 512-523.
14. Y.-K. Huang and L.-F. Lin, "Continuous within query in road networks," in *Proceedings of the 7th International Wireless Communications and Mobile Computing Conference*, 2011, pp. 1176-1181.
15. G. R. Hjaltason and H. Samet, "Incremental distance join algorithms for spatial databases," in *Proceedings of International Conference on ACM SIGMOD*, 1998, pp. 237-248.
16. T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient processing of spatial joins using r-trees," in *Proceedings of International Conference on ACM SIGMOD*, 1993, pp. 237-246.
17. N. Mamoulis and D. Papadias, "Multiway spatial joins," *ACM Transactions on Database Systems*, Vol. 26, 2001, pp. 424-475.
18. 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, Article No. 7.
19. 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.
20. C. Long, R. C.-W. Wong, K. Wand, and A. W.-C. Fu, "Collective spatial keyword queries: A distance owner-driven approach," in *Proceedings of International Conference on ACM SIGMOD*, 2013, pp. 689-700.
21. 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.
22. http://www.openstreetmap.org/.

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.