Processing Directional Continuous Queries for Mobile Objects on Road Networks

Chow-Sing Lin, Ci-Ruei Jiang

Department of Computer Science and Information Engineering, National University of Tainan, Taiwan
mikelin@mail.nutn.edu.tw, ha9mv8c@gmail.com

Abstract

Traditional location-based services for range queries or *k* nearest neighbor queries are omnidirectional, in which some query results are probably not interesting to a user because they are not on the way to destination. Due to the prevalence of guiding systems and the advances of electronic technologies and wireless communications, most mobile objects on roads move along with the pre-determined paths provided by path planning systems. With the concept of "by the way", there would be an increasing demand on queries for objects on their paths and ahead of them. In this paper, we define the issues of directional continuous query for mobile objects on road networks. To address this issue, we not only propose the system architecture of a road network but also provide approaches to determine the safe period in order to efficiently update a directional query result. We elaborate cases of when to exclude a mobile object from a query result and when to insert a mobile object into a query result. Experimental results show that the network bandwidth dominates the preference of the adaptation of the centralized or distributed directional mobile query processing.

Keywords: Range query, *k*NN query, Directional continuous query, Road network, Safe period

1 Introduction

With the rapid development of wireless communication networks and smart mobile devices, travelers now can access GPS localization services and wireless networks at any time. To date, several studies of query processing on road networks have been made on *k*-nearest neighbor queries [1-7] and range queries [8-14] based on the different requirements of travelers. To the best of our knowledge, these studies all focused on omnidirectional queries. According to our observation, however, a traveler usually query objects of interest based on the direction of travel, which indicates that the direction of traversals would influence the results of queries. In this situation, objects in the query result that are not on the way to destination would be considered as redundant.

Due to the prevalence of guiding systems and the advances of electronic technologies and wireless communications, apparently, most mobile objects on roads will move along with the pre-determined paths provided by their path planning systems. With the concept of "by the way", there would be an increasing demand on queries for objects on the travelling path and ahead of them. Such queries can be classified into two types, Range Query (RQ) and K-Nearest Neighbor Query(KNN). With range query, mobile objects within a certain range would be added into query result. On the contrast, only the *k*-nearest mobile objects would be added into query result. For example, what are the convenience stores within 3 km on the way home (RQ)? What are the three nearest taxies on the way to destination for carpooling (KNN)? What are the trucks belonging to the same fleet within 2 km on the way to the same destination (RQ)? Where are the two closest teammates in a cycling race (KNN)? To properly answer those questions, several challenges arise.

- How to represent the moving path of a mobile object?
- How to determine the moving path of a mobile object intersecting with another one?
- How to properly determine the result of a query for a query mobile object on the way to destination?
- How to compute the safe period [12-14] of updating a query result to reduce the frequency of position report?

On the other hand, queries can be processed in the central server (called centralized query processing) or in access points (called distributed query processing). Factors in an infrastructure, such computing power and network bandwidth, would favor one of these two processing types. In this paper, according to the types of query and processing, we discuss four directional query processings which are centralized directional range query (CDRQ), centralized directional KNN query (CDKQ), distributed directional range query (DDRQ), and distributed directional KNN query (DDKQ).

In this paper, we define the issues of these four

*Corresponding Author: Chow-Sing Lin; E-mail: mikelin@mail.nutn.edu.tw

DOI: 10.3966/160792642019012001009

directional continuous queries for mobile objects on a road network. To address these issues, we not only propose the system architecture of a road network but also design road block code to represent a moving path. How to determine one moving path intersected with another to produce proper query results is also discussed in the paper. In addition, we also provide approaches to determine a safe period in order to efficiently update a query result. We elaborate cases of when to exclude a mobile object from a query result and when to insert a mobile object into a query result.

To our best knowledge, this is the first research work which discusses the directional query processing for road networks with the concept of "by the way", which more commonly meets a user's daily needs. In this paper, we assume there is at least one access point (AP) located at each intersection, which may be premature in current infrastructure, but we truly believe it would be realized in future road networks.

Our experiments were deployed in centralized and distributed query processing schemes, and the experimental results show that when the bandwidth of a central server is high, the centralized directional continuous query processing is more efficient than the distributed one; on the other hand, the distributed query processing is better than the centralized one while the bandwidth of a central server is low. In addition, when the bandwidth of a central server is high, the centralized processing directional continuous queries has shorter average response time than the distributed one with the larger range in range query. When the value of *k* and the bandwidth of a central server are high, the centralized *k* Nearest Neighbor (*k*NN) query has shorter average response time than the distributed one.

The remainder of this paper is organized as follows. System architecture of road networks is described in Section 2. In Section 3, 4, 5, and 6, we describe the concept of directional continuous query and provide detailed discussion of how to efficiently update the query result by safe period in centralized range query, centralized *k*NN query, distributed range query, and distributed *k*NN query, respectively. The experimental results are presented in Section 7. Finally, Conclusions are drawn in Section 8.

2 System Architecture of Road Networks

In our system, a mobile object is equipped with GPS and its moving path to the destination is assumed to be pre-determined by a central server capable of path planning, such as Google Maps. There is an access point (or called roadside unit) at each intersection which is capable of communicating with mobile objects within a certain range by wireless communication. It is assumed that the wireless communication range for a road section can be equally covered by access points located at the both ends of a road section. Access points are interconnected by underground cables, which are formed as a meshed road network. An access point is responsible for monitoring mobile objects [15-16] within its service range, bookkeeping information of those mobile objects, processing queries, relaying communication messages between mobile objects and a central server, etc. The central server is wire connected to the meshed road network, and it is responsible for path planning, maintaining locations of mobile objects, processing query, etc. Figure 1 shows the system architecture of a road network [17-20].

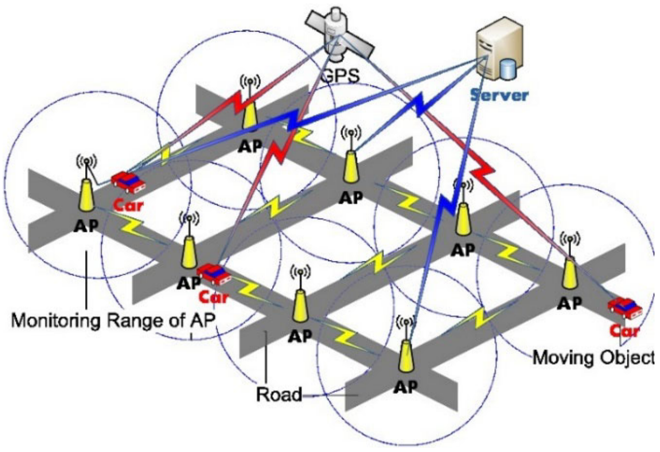
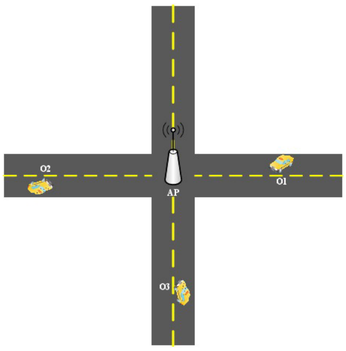


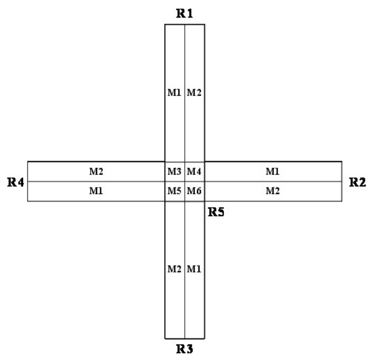
Figure 1. System architecture of a road network

In general, roads consist of a sequence of road sections and intersections [21-22]. To efficiently process moving paths of mobile objects in our system, we can divide roads into a number of "roadlet" which consists of an intersection and road sections connected to the intersection. An intersection may be 3- way, called T junction or fork; or 4-way intersection, called a crossroad; or more. In this paper, without losing generality, we assume all intersections on roads are 4-way. Figure 2(a) shows a road let with 4-way intersection. As shown in the figure, a roadlet can be divided into five regions which are the intersection (R5), the eastern road section (R2), the western road section (R4), the southern road section (R3), and the northern road section (R1).

The intersection R5 is further divided into four equal blocks which are upper right (M4), upper left (M3), lower right (M6), and lower left (M5). Each road section is also further divided into two blocks M1 and M2 based on driving directions. Figure 2(b) shows the representation of an intersection and four road sections of a roadlet. Assume that a roadlet is identified by the unique ID of its associated access point, each road block can be uniquely coded by sequentially concatenating the ID of the resided roadlet (access point), the region ID, and the block ID, which is called road block code, such as AP12R4M3. Given a pre-determined moving path provided by a route planning system, it now can be represented as a sequence of road block codes. Figure 3 shows an example of the



(a) Roadlet of 4-way intersection



(b) Representation of an intersection and five road sections

Figure 2. Representation of a roadlet

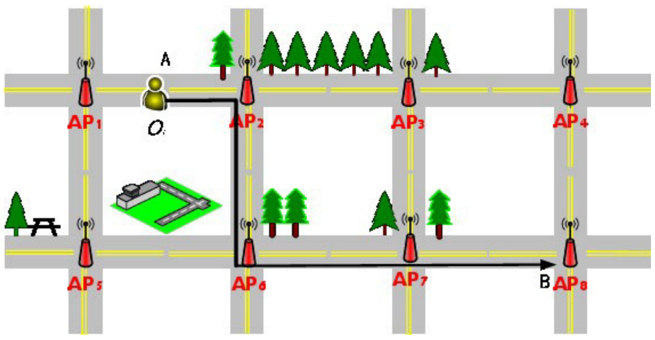


Figure 3. An example of moving path for a mobile object

path moving from point A to B for a mobile object, O_1. The road block code of this moving path is {AP1R2M2, AP2R4M1, AP2R5M5, AP2R3M2, AP6R1M1, AP6R5M3, AP6R5M5, AP6R5M6, AP6R2M2, AP7R4M1, AP7R5M5, AP7R5M6, AP7R2M2, AP8R4M1}. Note that the first element of a road block code represents the current location of a mobile object and it is accordingly updated as it is moving. The hierarchical structure of road blocks also favors the indexing of mobile objects by any kind of data structure used for spatial searching, such as R-tree. Figure 4 shows the indexing in R-tree for the three mobile objects in Figure 2(a). Note that literatures showed that R-TPR^{+-}Tree [23-24] might be more suitable for indexing mobile objects on road networks. However, our system architecture is independent of any spatial indexing methods and the discussion of the pros and cons of those indexing methods is beyond the scope of this paper.

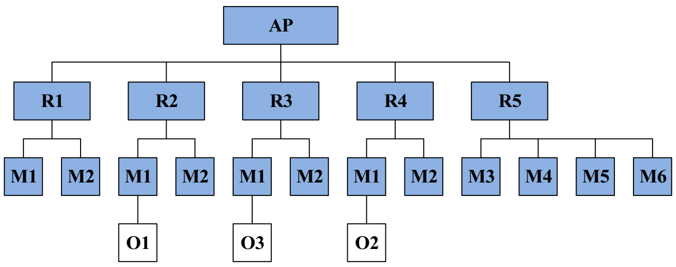


Figure 4. Indexing in R-tree for the three mobile objects in Figure 2(a)

3 Centralized Directional Continuous Range Query(CDRQ) Processing

Assume that there are *m* mobile objects on the roads, the road block code of the moving path for a mobile object O_i is denoted as C_i, $1 \le i \le m$, and the *k*-th element of his road block code is denoted as $C_{i,k}$. Given the moving path of a query mobile object *Q*, how many mobile objects with the same driving direction will appear in his query, such as range or *k*-Nearest Neighbors (*k*NN) query, on his way to destination? This type of query is called directional continuous query. To answer this kind of queries, we need to compare the road block code of the query mobile object against that of every mobile objects, says O_i, on the roads to find out if an intersection between the two road block codes exists, which leads to two cases. The first case is that there is no intersection between the road block code of *Q* and that of any mobile objects on the roads. That is, $C_Q \cap C_i = \varnothing$, $1 \le i \le m$. No mobile objects will become the query result on the way of the *Q* to destination, and therefore, there are no mobile objects to be monitored. The second case, there exists an intersection between the road block code of *Q* and that of any mobile object on the roads. That is, $C_Q \cap C_i \ne \varnothing$, $1 \le i \le m$. Some mobile objects will become the query result on the way of *Q* to destination. Assume that the $R_{potential}$ denotes the set of mobile objects whose road block codes intersect with the road block code of a query mobile object C_Q, $R_{potential}$ can be computed as follows,

$$R_{potential} = \{O_i \mid C_Q \cap C_i \ne \varnothing, 1 \le i \le m\} \qquad \textbf{(1)}$$

Mobile objects in Rpotential whose first element of the road block code belongs to C_Q may be immediately included into the query result, *R*, once the query is issued based on the criteria of the query type, such as the range for range query (e.g., 1km) or the value of *k* for *k*NN (e.g., *k*=5). Therefore, the initial query result, R_{init}, can be computed as follows.

$$R_{init} = \{O_i \mid \forall O_i \in R_{potential}, C_Q \cap C_{i,1} \neq \varnothing, q\_crit(O_i)\text{holds}\},$$

where *q_crit*() represents the criteria function of query type which can be range query or *k*NN query. In this case, *q_crit*() represents range query.

Initial, $R = R_{init}$. Assume that the set of mobile objects which may be included into R is denoted as $R_{candidate}$,

$$R_{candidate} = R_{potential} - R_{init}. \quad \textbf{(2)}$$

The update of query result R involves in excluding mobile objects from R and including mobile objects in $R_{candidate}$ into R, which will be discussed in the following section. Figure 5 shows an example of a query mobile object Q and mobile objects O_1, O_2, O_3, and O_4. As shown in the figure, Q is moving from location A to B, O_1 is moving from location C to D, O_2 is moving from location E to F, O_3 is moving from location G to H, O_4 is moving from location I to J. Table 1 shows the road block codes of these five mobile objects. Based on above equations, we can get $R_{potential} = \{O_1, O_3, O_4\}$, $R_{init} = \{O_1\}$ and $R_{candidate} = \{O_3, O_4\}$.

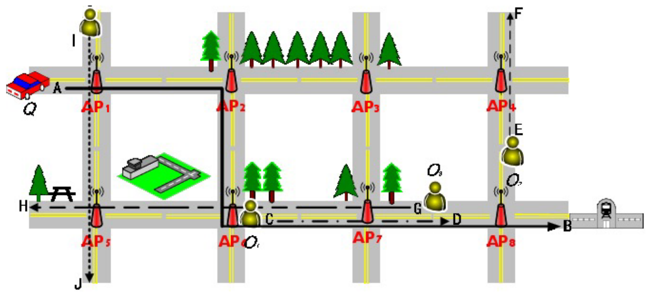


Figure 5. Moving paths of Q, O_1, O_2, O_3, and O_4

Table 1. The road block codes of mobile objects in Figure 5

Type	Path
Q	AP1R4M1, AP1R5M5, AP1R5M6, AP1R2M2, AP2R4M1, AP2R5M5, AP2R5M6, AP2R3M2, AP6R1M1, AP6R5M3, AP6R5M6, AP6R2M2, AP7R4M1, AP7R5M5, AP7R5M6, AP7R2M2, AP8R4M1, AP8R5M5, AP8R5M6, AP8R2M2
O_1	AP6R2M2, AP7R4M1, AP7R5M5, AP7R5M6, AP7R2M2, AP8R4M1
O_2	AP8R1M2, AP4R3M1, AP4R5M6, AP4R5M4, AP4R1M2
O_3	AP7R2M1, AP7R5M4, AP7R5M3, AP7R4M2, AP6R2M1, AP6R5M4, AP6R5M3, AP6R4M2, AP5R2M1, AP5R5M4, AP5R5M3, AP5R4M2
O_4	AP1R1M1, AP1R5M3, AP1R5M5, AP1R3M2, AP5R1M1, AP5R5M3, AP5R5M5, AP5R3M2

3.1 Update of Query Result

Once the initial query result is generated, the next question is when the query result R should be updated? This would be classified in two scenarios. First, R should be updated when mobile objects in R are no longer eligible for query result. That is, mobile objects which would be excluded from R are the ones deviating from the route of Q or getting out of its query range. Second, R should be updated when mobile objects in $R_{candidate}$ are eligible for query result. This would happen when a mobile object enters the moving path of Q or the query range. Intuitively, frequently updating the current locations of mobile objects would result in accurate query result at the cost of the consumption of network bandwidth and computation. To address this issue, we also apply the concept of safe period to reduce the frequency of location update for power saving in above two scenarios while updating R, which are discussed in the following sections.

3.2 Excluding Mobile Objects from Query Result

In this paper, we assume the speed of a mobile object O is bounded in $[V_O^{\min}, V_O^{\max}]$. When a mobile object in query result R being excluded can be classified into three cases, which are 1) the query mobile object Q catching up with a mobile object in R; 2) a mobile object in R moving out of the query range; 3) a mobile object in R deviating from the moving path of Q. Figure 6 shows these three possible exit points, denoted accordingly as $t_{catchup}$, t_{exit}, and $t_{deviate}$ for a mobile object O being excluded from the range query result.

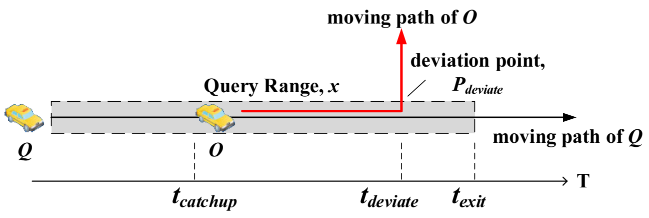


Figure 6. Three possible exit points for a mobile object O being excluded from the range query result

1. $t_{catchup}$: The shortest time that the query mobile object Q may catch up with a mobile object O. If $V_Q^{max} \leq V_O^{min}$, there is no way that Q would catch up with Q; otherwise, the equivalence of the catch up time, $t_{catchup}$, can be stated as

$$V_Q^{max} \times t_{actchup} = Dist(Q, O) + V_O^{min} \times t_{actchup}$$

and

$$t_{actchup} = Dist(Q, O) / (V_Q^{max} - V_O^{min}), \quad \textbf{(3)}$$

where $Dist(Q, O)$ denotes the current distance between Q and O.

2. t_{exit}: The shortest time that a mobile object O may move out of the query range x at the same driving direction. If $V_Q^{max} \leq V_O^{min}$, this case would not happen; otherwise, the equivalence of the exit time, t_{exit}, can be stated as

$$V_O^{max} \times t_{exit} = (x - Dist(Q,O)) + V_Q^{min} \times t_{exit}$$

and

$$t_{exit} = (x - Dist(Q,O)/(V_O^{max} - V_Q^{min})). \tag{4}$$

3. $t_{deviate}$: The shortest time that a mobile object O may deviate from the moving path of Q. The $t_{deviate}$ can be computed as, $t_{DEVIATE} = Dist(O,P_{deviate})/V_O^{max}$, where $P_{deviate}$ denotes the location of O deviating from the moving path of Q.

Let SP_O denotes the safe period of a mobile object O in R, and then SP_O = Min $\{t_{catchup}, t_{exit}, t_{deviate}\}$. Within the safe period, SP_O, the O being one of query result R remains valid, and there is no need for O to report its location to the central server. Once the safe period SP_O is elapsed, the O needs to report its location to the central server for updating the query result and compute a new safe period based on steps described above.

3.3 Inserting Mobile Objects into Query Result

For those who in the $R_{candidate}$, it is necessary to estimate when they are going to be included into R. Mobile objects in $R_{candidate}$ can be classified into two types. The first type includes mobile objects that are already on the moving path of Q but not within the query range. The first element of road block code of a mobile object, O, in this type must belong to road block code of the query mobile object, i.e., $C_{O,1} \in C_Q$. Figure 7 shows the case of a mobile object O ahead of a query mobile object Q but not within the query range. If $V_Q^{max} \le V_O^{min}$, the O would be definitely impossible to enter the query range and be included into R. In this case, the O should be immediately excluded from $R_{candidate}$. Otherwise, the equivalence of the shortest time of O being included into R, t_{enter}, can be stated as $Dist(Q,O) + V_O^{max} \times t_{enter} = x + V_Q^{max} \times t_{enter}$, and t_{enter} = $(Dist(Q,O) - x)/(-V_Q^{max} - V_O^{min})$. Note that if $t_{enter} \ge t_{deviate}$, a mobile object deviates from the moving path of Q before becoming the query result. In this case, again, this mobile object is excluded from $R_{candidate}$.

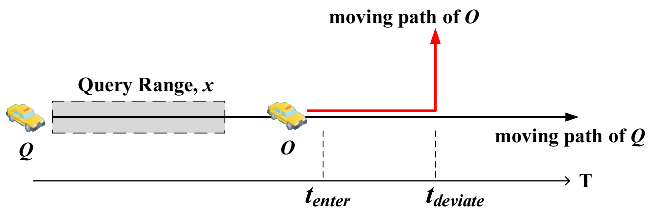


Figure 7. A mobile object O ahead of the query mobile object Q but not within the query range

The second type included mobile objects who are not currently on the moving path of Q but will be in the future. A mobile object in this type whose road block code C_O except the first element intersects with C_Q, i.e., $(C_O - C_{O,1}) \cap C_Q \ne \varnothing$. Figure 8 shows the case of a mobile object O will move into the travelling path of a query mobile object Q at the rendezvous point, P_{rend}. The shortest rendezvous time, t_{rend} can be computed as, $t_{rend} = Dist(O,P_{rend})/V_O^{max}$. Once the t_{rend} is elapsed, there are three possible cases, (1) the O is within the query range of Q; (2) the O is ahead of Q but not within the query range; (3) the O is behind Q. Cases 1 and 2 are identical to the cases of Figure 6 and Figure 7, respectively. Figure 9 shows case 3 is a mobile object O behind a query mobile object Q. If $V_O^{max} \le V_Q^{min}$, there is no way that the O can be included into the query result, and it is then excluded from $R_{candidate}$; otherwise, the equivalence of the shortest time to have O become the query result, t_{arrive}, would be $V_O^{max} \times t_{arrive} = V_Q^{min} \times t_{arrive} + Dist(Q,O)$, and t_{arrive} = $Dist(Q,O)/(V_O^{max} - V_Q^{min})$. Again, if $t_{arrive} \ge t_{deviate}$, the O is immediately excluded from $R_{candidate}$. Algorithm 1 shows the algorithm of CDRQ.

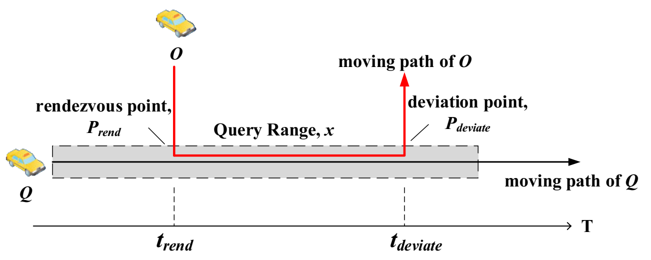


Figure 8. A mobile object O moving into the traveling path of a query mobile object Q

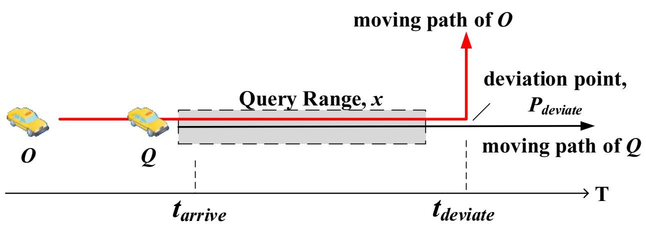


Figure 9. A mobile object O behind a query mobile object Q

4 Centralized Directional Continuous *k*NN Query(CDKQ) Processing

In CDKQ, we need to find the k-nearest objects ahead of the mobile object Q on its moving path. The *q_crit*() in the definition of R_{init} now represents *k*NN query. Note that if the number of the returned mobile objects is less than k, the query is failed. For an example of Figure 5 and k equal to one, we can get $R_{potential} = \{O_1, O_3, O_4\}$, $R_{init} = \{O_1\}$ and $R_{candidate} = \{O_3, O_4\}$.

```
Algorithm 1: Centralized Directional Continuous Range Query(CDRQ)

Input: The query mobile object, $Q$, the range of the
       range query
Output: The result of the range query, $R$

Generate Initial Results
Find the list of involved APs, $I$
$\forall P_i, P_i \in I$
if $O \in FRB$ then
    $R = R \cup O$
else if $O \in PRB$, calculate the distance d, between
  O and $P_1$ then
    if $d \leq x$ then
        $R = R \cup O$
$\forall O, O \in R$
if $O \in FRB$ then
    case1:
    if $P_{deviate}$ is in the resided road block then
        calculate $t_{deviate}$
        $SP_O = t_{deviate}$
    else
        calculate $t_{left}$
        $SP_O = t_{left}$
else if $O \in frontPRB$ then
    if $V_Q^{\max} \leq V_O^{\min}$ then
        similar to case1
    else
        if $P_{deviate}$ is in the resided road block then
            calculate $t_{deviate}, t_{left}$
            $SP_O = \mathrm{Min}\{t_{deviate}, t_{left}\}$
        else
            calculate $t_{left}, t_{catchup}$
            $SP_O = \mathrm{Min}\{t_{catchup}, t_{left}\}$
else if $O \in$ rear PRB then
    if $V_O^{\max} \leq V_Q^{\min}$ then
        similar to case1
    else
        calculate $t_{deviate}, t_{left}, t_{moveout}$
        $SP_O = \mathrm{Min}\{t_{deviate}, t_{moveout}, t_{left}\}$
Update Query Result
$\exists O, SP_O$ equals 0
if $O \in R$ then
    recalculate $SP_O$ according to the previous steps
else
    if $R$ would be affected by $O$ then
        $R = R \cap O$
        calculate $SP_O$ according to the previous steps
```

In order to reduce the frequency of updating query result, again we also adopt the concept of safe period. The update of *k*NN query results is same as that of range query which can be classified into two possible cases, which are (1) excluding a mobile object from the query result *R*; (2) inserting a mobile object into the query result *R*. Note that in *k*NN query excluding a mobile object from *R* will trigger inserting *R* a mobile object, which is just ahead of the *k*th mobile object in *R*, to maintain |*R*| equal to *k*. On the other hand, inserting a mobile object into *R* will automatically remove the last mobile object from *R*. The computation of $t_{catchup}$, $t_{deviate}$, t_{enter}, and t_{rend} in CDKQ is very similar to that in CDRQ and to save space, there are no further discussions in these issues. Algorithm 2 shows the algorithm of CDKQ.

```
Algorithm 2: Centralized Directional Continuous k-Nearest Neighbor Query(CDKQ)

Input: The query mobile object, $Q$, the value of the $k$
Output: The result of the range query, $R$

Generate Initial Results
$\forall mobileobjects, O_i$
if $C_Q \cap C_{i,1} \neq \emptyset$ then
    Add $O_i$ into the set of initial result, $R_{init}$
    Add $O_i$ into the set of potential result, $R_{potential}$
else if $C_Q \cap C_i \neq \emptyset$ then
    Add $O_i$ into the set of potential result, $R_{potential}$
The set of candidate result,
  $R_{candidate} = R_{potential} - R_{init}$
if the number of $R_{init} \geq k$ then
    $\forall O, O \in R_{init}$
    if the number of $R < k$ then
        $R = R \cup O$
    else
        $R_{init} = R_{init} - O$
        $R_{candidate} = R_{candidate} \cup O$
else
    $R = R_{init}$
    $\forall O, O \in R_{init}$
    Calculate $t_{deviate}, t_{catchup}$
    $SP_O = \mathrm{Min}\{t_{catchup}, t_{deviate}\}$
    $\forall O, O \in R_{candidate}$
    Calculate $t_{enter}, t_{rend}, t_{arrive}$
    $SP_O = \mathrm{Min}\{t_{enter}, t_{rend}, t_{arrive}\}$

Update Query Result
$\exists O, SP_O$ equal 0, $O \in R \cup R_{candidate}$
if $O \in R$ then
    if $O \notin$ the first k mobile objects then
        $R = R - O$
        $R = R \cup$ the next mobile object
        Calculate $t_{deviate}, t_{catchup}$ of the next object
        $SP_O = \mathrm{Min}\{t_{catchup}, t_{deviate}\}$
    else
        Calculate $t_{deviate}, t_{catchup}$
        $SP_O = \mathrm{Min}\{t_{catchup}, t_{deviate}\}$
else if $O \in R_{candidate}$ then
    if $O \in$ the first k mobile objects then
        $R = R$ - the original $k - th$ mobile object
        $R_{candidate} = R_{candidate} - O$
        $R = R \cup O$
        Calculate $t_{deviate}, t_{catchup}$
        $SP_O = \mathrm{Min}\{t_{catchup} t_{deviate}\}$
    else
        Calculate $t_{enter}, t_{rend}, t_{arrive}$
        $SP_O = \mathrm{Min}\{t_{enter}, t_{rend}, t_{arrive}\}$
```

5 Distributed Directional Continuous Range Query (DDRQ) Processing

In DDRQ, a mobile object is assumed to have full knowledge about the travelling route, i.e., its road block code. An AP maintains an R-tree as shown in Figure 4 to index mobile objects in its monitoring range. In DDRQ, queries are processed only in APs without the intervention of a central server. When a mobile object, O_i, issues a range query, which may contain $\{O_i, C_i, x\}$, its resided AP would receive and then process the query. If the query can be satisfied, the resided AP would directly response the query result to the mobile object; otherwise, the remaining query will be forwarded to the next AP on its moving path until the query is fully answered. Such a concept is called "query forwarding" shown in Figure 10. Apparently, each involved AP would be also responsible for returning the query result to the originated AP and finally the resided AP provides the collected query result to the mobile object.

In DDRQ, given the location of a query mobile object, assume that *P* represents the list of the APs possible involving in query processing, which would orderly include its resided AP and APs ahead of the resided AP on the moving path. Let P_i denote the *i*th AP in *P* and *I* denote the APs responsible for answering the query. The steps of determining *I* are described as follows.

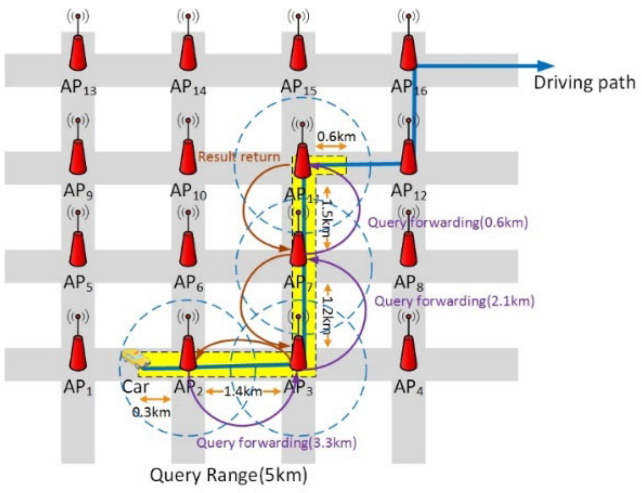


Figure 10. The query forwarding in DDRQ

Step 1. i =1 and P_i is inserted into I. Determine the location of query object Q with respect to the resided AP. If Q is behind the resided AP, go to Step 2; otherwise, go to Step 3.

Step 2. If $x > Dist(P_i, P_{i+1})/2 + Dist(Q, P_i)$, $x \leftarrow x - (Dist(P_i, P_{i+1})/2 + Dist(Q, P_i))$, go to Step 4.

Step 3. If $x > Dist(P_i, P_{i+1})/2 - Dist(Q, P_i)$, $x \leftarrow x - (Dist(P_i, P_{i+1})/2 + Dist(Q, P_i))$.

Step 4. $i \leftarrow i+1$. P_i is inserted into I. If $x > Dist(P_i, P_{i+1})/2 + Dist(P_{i+1}, P_{i+2})$, $x \leftarrow x - Dist(P_i, P_{i+1})/2 - Dist(P_{i+1}, P_{i+2})$ go to Step 4.

Each AP would maintain a *query table* whose attributes may include object id, query type, query quantity (query range or k), involved road blocks, and query result, to trace active queries processed by the AP. When a query object is handed off to the next AP, the query issued by it should be removed from the query table in the previous AP. When to exclude/include mobile objects from/to query result for DDRQ is discussed in the following sections.

5.1 Excluding Mobile Objects from Query Result

When a query is issued, a series of road blocks starting from the current road block of the query object are involved in answering the query, which can be divided into two sets of road blocks called *partial road block (PRB)* and *full road block (FRB)*. Generally, the first and last road block, called the *front* and *rear* partial road block respectively, may become the member of *PRB* where only some objects in these two road blocks may be the query result, and other road blocks are in the set of *FRB*. Figure 11 shows an example of the partial and full road blocks where AP6R2M2 and AP7R3M2 belong to *PRB*; AP6R4M1 and AP7R4M5 belong to *FRB*. Note that road blocks in *PRB* or *FRB* are accordingly updated along with the moving of the query object.

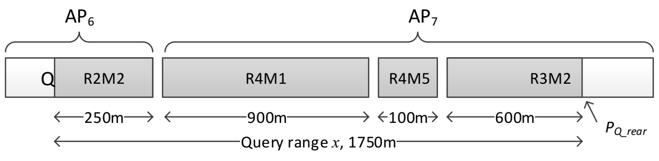


Figure 11. The partial and full road blocks

Once an initial query result is created, it should be updated periodically by applying safe period to avoid excessive location reporting of mobile objects, which can be categorized into three cases:

1. For each mobile object in *FRB*, a mobile object should report its location to the resided AP once its safe period straying its current road block this query is expired. If the deviation point, $P_{deviate}$, is in the resided road block, the safe period can be easily determined by $t_{deviate}$. If not, its safe period can be calculated by $t_{left} = Dist(O, E)/V_O^{MAX}$, where E denotes the end point of the resided road block. Figure 12 shows an example of safe periods of two objects in FRB.

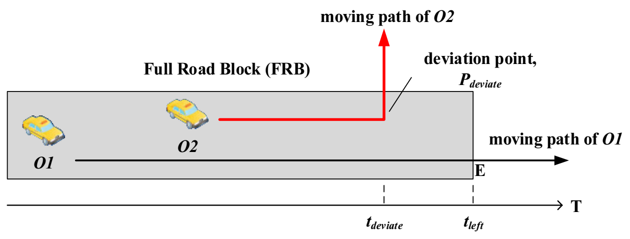


Figure 12. Safe periods of two objects in FRB

2. For each mobile object in the front partial road block, if $V_Q^{MAX} \le V_O^{min}$, the safe period can be determined similar to the case of mobile objects in *FRB*; otherwise, if its $P_{deviate}$ is not in the resided road block, we compute t_{lefe} and $t_{catchup}$. If the $t_{catchup}$ is less than t_{left}, the safe period is $t_{catchup}$.; otherwise is t_{left}. On the other hand, if its $P_{deviate}$ is in the resided road block, similarly, if the $t_{deviate}$ is less than t_{left}, the safe period is $t_{deviate}$; otherwise is t_{left}. The safe period would be the minimum value of $t_{catchup}$, t_{left}, and $t_{deviate}$. Figure 13 shows an example of safe periods of two objects in front PRB.

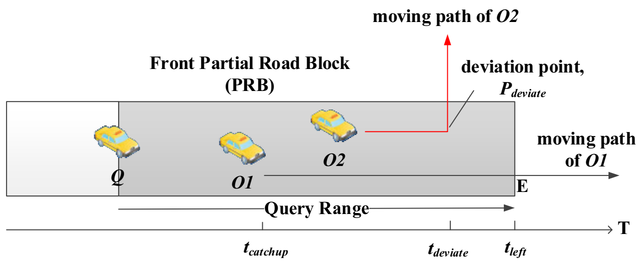


Figure 13. Safe periods of two objects in front PRB

3. For each mobile object in the rear partial road block, if $V_O^{MAX} \le V_Q^{MIN}$, the safe period is $t_{deviate}$ if its deviation point is in this road block; the safe period is

t_{left} if not. On the other hand, if $V_O^{MAX} > V_Q^{MIN}$, we first compute the time that the mobile object moves out of the query range, $t_{moveout} = Dist(O, P_{Q\_rear})/(V_O^{MAX} - V_Q^{MIN})$. The safe period would be the minimum value of $t_{moveout}$, t_{left}, and $t_{deviate}$. Figure 14 shows an example of safe periods of two objects in rear PRB.

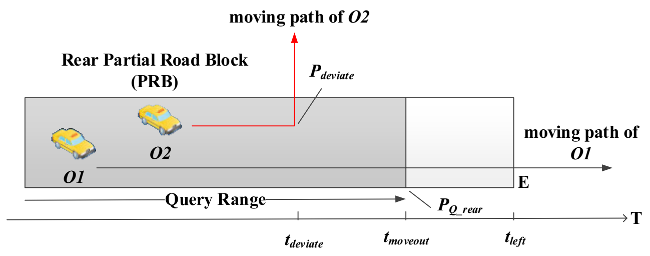


Figure 14. Safe periods of two objects in rear PRB

Note that once a query object Q moves into a new road block, this road block immediately becomes the first partial road block. On the other hand, as Q moves, the rear partial road block would become a full road block. If the type of a road block changes, the safe period of objects resided in the road block should be updated accordingly if necessary.

5.2 Inserting Mobile Objects into Query Result

In DDRQ, inserting mobile objects into existing query results can be triggered by handing off messages received by destination APs. That is, when a mobile object moves into a new road block, the destination AP would proactively be notified its arrival and the results of queries listed in the destination AP should be updated accordingly. The safe period of moving into a new road block for a mobile object can be easily obtained by t_{left}.

Furthermore, objects possibly being inserted into a query result would be those who are in front or rear PRB but not inside the query range, similar to cases shown in Figure 7 and Figure 9. For those objects ahead and behind the query range, their safe periods to become the query result can be easily calculated by t_{enter} and t_{left}, respectively. Algorithm 3 shows the algorithm of DDRQ.

Algorithm 3: Distributed Directional Continuous Range Query(DDRQ)

```
Input: The query mobile object, $Q$, the range of the
       range query, $x$
Output: The result of the range query, $R$

Generate Initial Results
Find the list of involved APs, $I$
$\forall P_i, P_i \in I$
if $O \in FRB$ then
    $R = R \cup O$
else if $O \in PRB$, calculate the distance d, between
  $O$ and $P_1$ then
    if $d \leq x$ then
        $R = R \cup O$
$\forall O, O \in R$
if $O \in FRB$ then
    case1:
    if $P_{deviate}$ is in the resided road block then
        calculate $t_{deviate}$
        $SP_O = t_{deviate}$
    else
        calculate $t_{left}$
        $SP_O = t_{left}$
else if $O \in frontPRB$ then
    if $V_Q^{max} \leq V_O^{min}$ then
        similar to case1
    else
        if $P_{deviate}$ is in the resided road block then
            calculate $t_{deviate}, t_{left}$
            $SP_O = \mathrm{Min}\{t_{deviate}, t_{left}\}$
        else
            calculate $t_{left}, t_{catchup}$
            $SP_O = \mathrm{Min}\{t_{catchup}, t_{left}\}$
else if $O \in rearPRB$ then
    if $V_O^{max} \leq V_Q^{min}$ then
        similar to case1
    else
        calculate $t_{deviate}, t_{left}, t_{moveout}$
        $SP_O = \mathrm{Min}\{t_{deviate}, t_{moveout}, t_{left}\}$
Update Query Result
$\exists O, SP_O$ equals 0
if $O \in R$ then
    if $O$ is still in $R$ then
        recalculate $SP_O$ according to the previous
          steps
    else
        $R = R - O$
else
    if $R$ would be affected by $O$ then
        $R = R \cup O$
        calculate $SP_O$ according to the previous steps
```

6 Distributed Directional Continuous *k*NN Query (DDKQ) Processing

In DDKQ, a mobile object Q issues a kNN query to its resided AP, says $k = 8$. If the query can be satisfied by the resided AP, the query result would be answered immediately; otherwise, the remaining query, says $k = 6$, would be forwarded to the next AP on the moving path until the query could be totally satisfied, and the query result would be then routed back to the originated AP, similar to DDRQ. Once the initial query result is generated, the next question is when to update the result. Again, we apply the concept of safe period to effectively reduce the frequency of position reporting of mobile objects.

Like DDRQ, APs and road blocks responsible for answering the query can be more than one. Figure 15 shows an example of DDKO, k=8. The involved APs can be classified as front PRB, FRB and rear PRB. The determination of safe periods of excluding mobile objects from query result is same as the one in DDRQ. Note that in DDKQ, each exclusion would trigger the insertion of the mobile object ahead of the last mobile object in query result (the kth) into query result which is handled by the AP of rear PRB. On the other hand, the determination of safe periods of insertion mobile objects into query result is again same as the one in DDRQ. Similarly, each insertion would trigger the exclusion of the the last mobile object in query result (the kth) from query result which is handled by the AP of rear PRB. Algorithm 4 shows the algorithm of DDKQ.

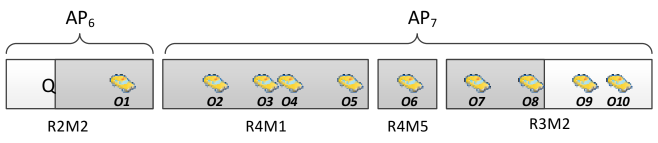


Figure 15. An example of DDKQ, *k*=8

```
Algorithm 4: Distributed Directional Continuous kNN Query(DDKQ)
Input: The query mobile object, $Q$, the value of $k$
Output: The result of the range query, $R$

Generate Initial Results
Find the list of involved APs, $I$
$\forall P_i, P_i \in I$
$R = R \cup O$ until the number of $R$ equals $k$
  $\forall O, O \in R$
if $O \in FRB$ then
    case1:
    if $P_{deviate}$ is in the resided road block then
        calculate $t_{deviate}$
        $SP_O = t_{deviate}$
    else
        calculate $t_{left}$
        $SP_O = t_{left}$
else if $O \in$ front PRB then
    if $V_Q^{\max} \leq V_O^{\min}$ then
        similar to case1
    else
        if $P_{deviate}$ is in the resided road block then
            calculate $t_{deviate}, t_{left}$
            $SP_O = \text{Min}\{t_{deviate}, t_{left}\}$
        else
            calculate $t_{left}, t_{catchup}$
            $SP_O = \text{Min}\{t_{catchup}, t_{left}\}$
else if $O \in rear PRB$ then
    if $V_O^{\max} \leq V_Q^{\min}$ then
        similar to case1
    else
        calculate $t_{deviate}, t_{left}, t_{moveout}$
        $SP_O = \text{Min}\{t_{deviate}, t_{moveout}, t_{left}\}$
Update Query Result
$\exists O, SP_O$ equals 0
if $O \in R$ then
    if $O$ is still in $R$ and $O \in$ first $k$ objects then
        recalculate $SP_O$ according to the previous
          steps
    else
        $R = R - O$
        find next $O$
else
    if $R$ would be affected by $O$ and $O \in$ the first $k$
      objects then
        $R = R \cup O - O_k$
        calculate $SP_O$ according to the previous steps
```

7 Experimental Results

Figure 16 shows the map of 1.6km×1.6km used in our experiments, which is the San Min district of Kaohsiung city in Taiwan with the center of latitude and longitude, (120.308342, 22.639607). The central server for the experiments is equipped with an Intel Core i7-4970, 4GB RAM, and running on 32-bit Windows 7. In our experiments, mobile objects were randomly scattered on the map. The mobility of a mobile object followed Random Waypoint Model [25-26]. The path planning was done by Google Maps [27]. Table 2 shows the default variables of the parameters of experiments. The bandwidths of a central server and an AP were 1000KBps and 10MBps. In addition, there was a dedicated bandwidth of 10MBps between two APs. The message packet size was 512KBytes [28]. The default value of *k* was 8 and the default range of a query was 1000m.

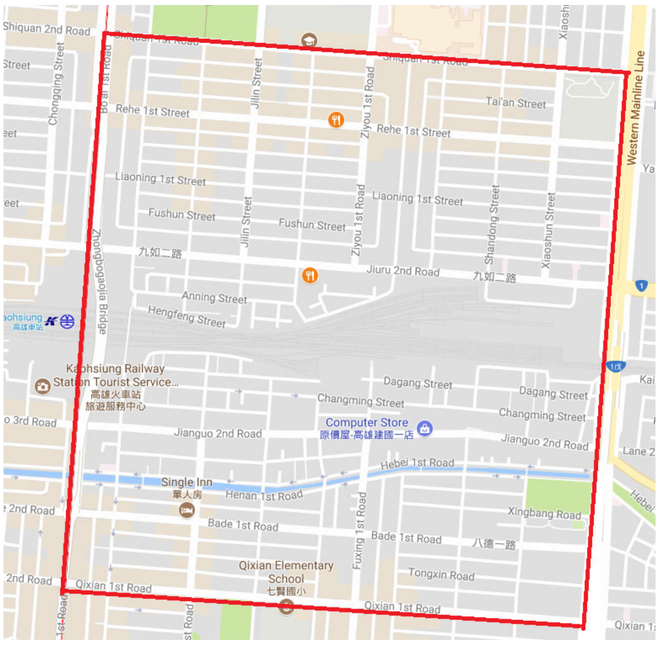


Figure 16. The map for experiments (San Min district, Kaohsiung City)

Table 2. The initial variables of the experimental environment

Variables	Value
Bandwidth of a central server	1000 Kbps
Bandwidth of and AP	10 Mbps
Bandwidth between two APs	10 Mbps
Message packet size	512 Kbytes
k	8
Range	1000 m

$$T_{centralized} = \frac{S_Q}{TB_{ap}} + \frac{S_Q}{TB_{srv}} + T_{srv} + \frac{S_R}{TB_{srv}} + \frac{S_R}{TB_{ap}} \quad (3)$$

We use Eq. 3 to calculate the response time of centralized query processing, $T_{centralized}$, where S_Q denotes the message packet size of a query mobile object Q. S_R denotes the return message packet size of a mobile object, S_O denotes the request message packet size of a mobile object O, T_{srv} denotes the process time of a central server, TB_{ap} denotes the bandwidth between an AP and a mobile object, and TB_{srv} denotes the bandwidth of a central server. When a mobile object issues a query, this query request first is delivered to the resided AP. Then, the AP passes the query to the central server. After finishing the query processing, the server returns the query result to the originated AP. Finally, the AP delivers the query result to the query mobile object.

Similarly, we use Eq. 4 to calculate the response time of distributed query processing, $T_{distributed}$, where TB_{ap-ap} denotes the dedicated bandwidth between two

APs, i denotes the number of APs involved in query processing, SR_p denotes the number of mobile objects satisfied with the query condition in the pth AP on the moving path, and T_{ap} denotes the query processing time of involved APs.

$$T_{distributed} = \frac{S_Q}{TB_{ap}} + S_O \times \sum_{p=1}^{i} X_i \frac{SR_p}{TB_{ap}} + T_{ap} + \frac{S_Q \times (i-1)}{TB_{ap-ap}} + \sum_{p=2}^{i} \frac{S_O^p \times (i-1)}{TB_{ap-ap}} \quad \textbf{(4)}$$

Figure 17 shows the average response time with respect to the various ranges for CDRQ and DDRQ. In this experiment, the bandwidth of a central server was 512KBps, and other parameters of the experiments are same as those showing in Table 2. As the query range increases, the number of the query results also rises, inevitably leading to the increase of the loading of computation and the number of message packets. As a result, the average response time of range query increases along with the increase of the range. As shown in the figure, with the query range 100m or less, the average response time of CDRQ is 15% longer than those of DDRQ under different query ranges. However, the upward trend of DDRQ is more noticeable than that of CDRQ, which is 5% higher than the latter. Because as the query range increases, the time of computing the distance between the query mobile object and the mobile objects whose moving paths overlap that of the query mobile object in CDRQ also increases, and the time of delivering the messages in DDRQ also increases. It is shown that the DDRQ rises more significantly than the CDRQ in our experiment. In addition, the lower value of initial bandwidth of a central server leads the longer average response time. When the query range is 322m which is calculated by interpolation method, the average response times of both are the same. With the query range of 500m, the average response time of CDRQ is shorter than that of DDRQ. On average, when the query range is between 500m and 1500m, the average response time of CDRQ is 19% shorter than those of DDRQ, and the upward change of DDRQ is 8% higher than that of CDRQ. In general, the effect of the query range is not obvious in CDRQ, but the growth of messages transferring time is significant in DDRQ. Clearly, DDRQ outperforms CDRQ only in small range of query.

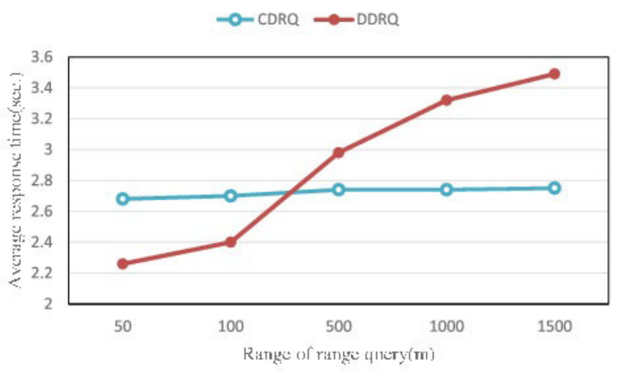


Figure 17. Effect of the range on CDRQ and DDRQ

Figure 18 illustrates the average response time with respect to the various values of k for CDKQ and DDKQ. In this experiment, the bandwidth of a central server was 1500KBps, the bandwidth between APs was 7.5MBps. As the value of k increases, again the number of query results also rises, undoubtedly causing the increase of loading of computation and the number of message packets. As a result, the average response time of kNN query increases along with the increase of the value of k. As shown in the figure, with

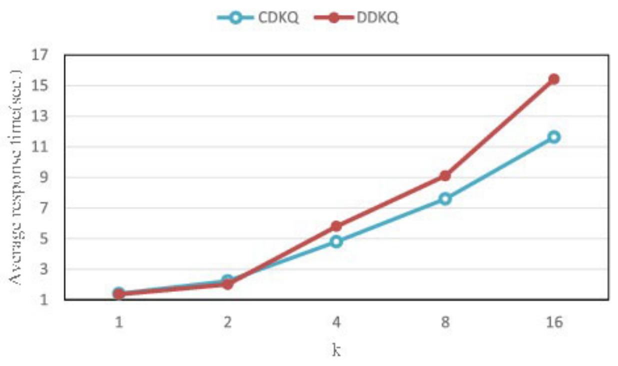


Figure 18. Effect of the value of k on CDKQ and DDKQ

the value of k less than or equal to 2, the average response time of DDKQ is 8% shorter than that of CDKQ. However, the upward trend of DDKQ is more noticeable than that of CDKQ, which is 12% higher than the latter. In CDKQ, we must first compute the distance between the query mobile object and the mobile objects whose moving paths overlap that of the query mobile object, and sort these overlapped mobile objects by the distance from the query object. The k-nearest mobile objects is then added into query result. On the other hand, in DDKQ, multiple APs may involve in answering the query result. With the small value of k, the query would not run through many APs so that the average response times of CDKQ are longer than those of DDKQ. When k is 3, both of the average response time is almost the same. When k equals to or larger than 4, the average response times of CDKQ is shorter than that of DDKQ because the number of involved APs increases along with the value of k. Generally, the difference of the average response times between two methods increases along with the increase of k. Obviously, the value of k affects the average response time more in DDKQ than in CDKQ. A larger value of k would definitely not favor CDKQ.

Figure 19 indicates the average response time with respect to the various bandwidths of a central server for CDRQ and DDRQ. In this experiment the bandwidth of a central server was 512KBps. Overall, as the bandwidth of a central server increases, the average

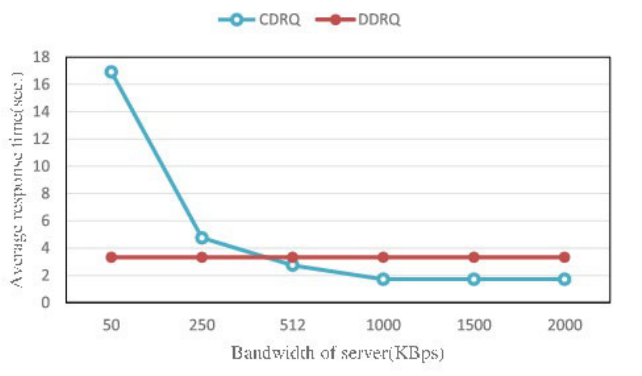


Figure 19. Effect of the bandwidth of a central server on range query

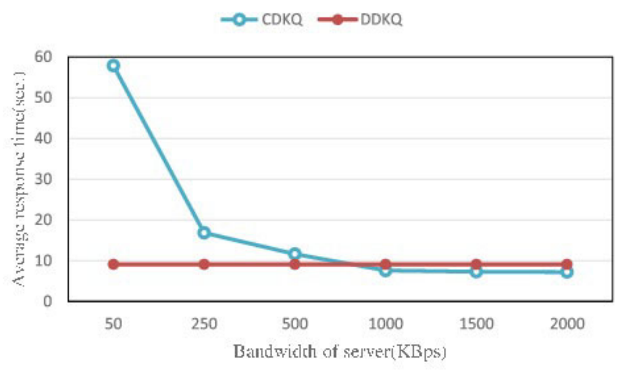


Figure 20. Effect of the bandwidth of a central server on *k*NN query

response time of CDRQ reduces drastically, and that of DDRQ is almost unaffected. In CDRQ, most of query processing is done by a central server and then the query result is passed to the query mobile object by interconnection network so that the bandwidth of the server significantly affects the response times. On the contrast, with DDRQ queries are processed locally by APs, and the central server is hardly involved in query processing. With the bandwidth of a central server 250KBps or less, CDRQ spends more time on query processing, which is 3.4 times more than that of DDRQ. As the bandwidth of the server is 326KBps, the average response times of both are the same. When the bandwidth of a central server reaches 512KBps, the average response time of CDRQ becomes shorter than that of DDRQ. With the bandwidth of the server reaches 1000KBps and beyond, the average response time of CDRQ is almost the same, where the time spending on delivering messages is negligible compared to the computation time.

With the bandwidth of a central server was 1500KBps and the bandwidth between APs was 7.5MBps, In Figure 20 shows the effect of the bandwidth of a central server on CDKQ and DDKQ similar to CDRQ and DDRQ. As the bandwidth of a central server increases, the average response time of CDKQ reduces dramatically, and that of DDKQ again does not change noticeably. With the bandwidth of a central server 500KBps or less, CDKQ spends more time on query processing, which is 3.2 times more than that of DDKQ. As the bandwidth of a central server is increased to 688KBps, the average response times of both are the same. When the bandwidth of a central server reaches 1000KBps, the average response time of CDRQ is shorter than that of DDRQ. Again, with the bandwidth of the server reaches 1000KBps and beyond, the average response time of CDKQ is almost the same, where the time spending on delivering messages is negligible compared to the computation time.

Figure 21 shows the average number of contacted APs with respect to the various query ranges for DDRQ and the various values of *k* for DDKQ. The figure shows that as the query range of DDRQ increases, the average number of contacted APs also

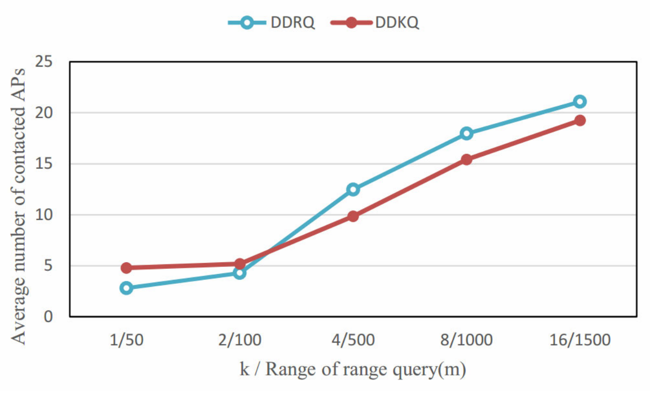


Figure 21. Effect of the average number of contacted APs in distributed query processing

increases, leading to the increase of the delivering time of messages and then the time of query processing shown in Figure 17. Similarly, as shown in the figure, the average number of contacted APs increases along with the increase of the value of *k*, leading to the increase of the query processing time in DDKQ shown in Figure 18.

8 Conclusion

In this paper, we address the issue of directional continuous query for mobile objects on road networks. We not only propose the system architecture of a road network but also provide approaches to determine the safe period in order to efficiently update the query result. We elaborate cases of when to exclude a mobile object from a query result and when to insert a mobile object into a query result. The results of our experiments show that the centralized directional continuous query processing has shorter average response time than the distributed one with the higher bandwidth of a central server. In contrast, when the bandwidth of a central server is low, the distributed directional continuous query processing is more efficient. In general, finding the most appropriate method will depend on network bandwidths and the type of mobile queries.

In the future, we plan to extend our research work to providing query services on intelligent transportation networks, or even for self-driving cars. Queries can be issued not only by humans but also by vehicles. As the intelligent infrastructure of transportations becomes more complete, realizing such services would surely become more feasible.

Acknowledgements

This research work was partially supported by Ministry of Science and Technology, Taiwan, under grand number 104-2221-E-024-002 and 103-2815-C-024-025-E. We also want to thank Ms. Tiffany Lin for her professional proofreading.

References

[1] U. Demiryurek, F. B. Kashani, C. Shahabi, Efficient Continuous Nearest Neighbor Query in Spatial Networks Using Euclidean Restriction, *Advances in Spatial and Temporal Databases*, Aalborg, Denmark, 2009, pp. 25-43.

[2] H. Samet, J. Sankaranarayanan, H. Albotzi, Scalable Network Distance Browsing in Spatial Databases, *Proceedings of the 2008 Association for Computing Machinery Special Interest Group on Management Of Data international conference on Management of Data*, Vancouver, Canada, 2008, pp. 43-54.

[3] K. C. K. Lee, W.-C. Lee, B. Zheng, Fast Object Search on Road Networks, *Proceedings of 12th International Conference on Extending Database Technology: Advances in Database Technology*, Saint Petersburg, Russia, 2009, pp. 1018-1029.

[4] H. Hu, D. L. Lee, V. C. S. Lee, Distance Indexing on Road Networks, *Proceedings of the 32nd International Conference on Very Large Data Bases*, Seoul, Korea, 2006, pp. 894-905.

[5] W. Wu, W. Guo, K.-L. Tan, Distributed Processing of Moving k-nearest Neighbor Query on Moving Objects, *International Conference on Data Engineering*, Istanbul, Turkey, 2007, pp. 1116-1125.

[6] G. S. Iwerks, H. Samet, K. Smith, Continuous k-nearest Neighbor Queries for Continuously Moving Points with Updates, *Proceedings of the 29th International Conference on Very Large Data Bases-Volume 29*, *Very Large Data Bases Endowment*, Berlin, Germany, 2003, pp. 512-523.

[7] T. P. Nghiem, A. B. Waluyo, D. Taniar, A Pure Peer-to-peer Approach for kNN Query Processing in Mobile Ad Hoc Networks, *Personal and Ubiquitous Computing*, Vol. 17, No. 5, pp. 973-985, June, 2013.

[8] H. Wang, R. Zimmermann, W. Ku, Distributed Continuous Range Query Processing on Moving Objects, *Proceedings of Database and Expert Systems Applications*, Krakow, Poland, 2006, pp. 655-665.

[9] B. Gedik, L. Liu, Mobieyes: A Distributed Location Monitoring Service Using Moving Location Queries, *Institute of Electrical and Electronics Engineers Transaction on Mobile Computing*, Vol. 5, No. 10, pp. 1384-1402, October, 2006.

[10] X. Wang, W. Wang, Continuous Expansion: Efficient Processing of Continuous Range Monitoring in Mobile Environments, *Database Systems for Advanced Applications*, Singapore, 2006, pp. 890-899.

[11] D. V. Kalashnikov, S. Prabhakar, W. G. Aref, S. E. Hambrusch, Efficient Evaluation of Continuous Range Queries on Moving Objects, *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, Aix en Provence, France, 2002, pp. 731-740.

[12] J. Lee, H. Jung, H. Y. Youn, U. M. Kim. SPQI: An Efficient Continuous Range Query Indexing Structure for a Mobile Environment, *Korean Institute of Information Scientists and Engineers Transactions on Computing Practices*, Vol. 21, No. 1, pp. 70-75, January, 2015.

[13] D. Stojanovic, S. Djordjevic-Kajan, A. N. Papadopoulos, A. Nanopoulos, Continuous Range Query Processing for Network Constrained Mobile Objects, *International Conference on Enterprise Information Systems*, Paphos, Cyprus, 2006, pp. 63-70.

[14] T. T. Do, K. A. Hua, C.-S. Lin, ExtRange: Continuous Moving Range Queries in Mobile Peer-to-Peer Networks, *Mobile Data Management: Systems, Services and Middleware, 2009 Tenth International Conference on Institute of Electrical and Electronics Engineers*, Taipei, Taiwan, 2009, pp. 317-322.

[15] Y.-L. Hsueh, R. Zimmermann, W.-S. Ku, Efficient Location Updates for Continuous Queries over Moving Objects, *Journal of Computer Science and Technology*, Vol. 25, No. 3, pp. 415-430, May, 2010.

[16] M. A. Cheema, L. Brankovic, W. Z. X. Lin, W. Wang, Continuous Monitoring of Distance Based Range Queries, *Institute of Electrical and Electronics Engineers Transaction on Knowledge and Data Engineering*, Vol. 23, No. 8, pp. 1182-1199, August, 2011.

[17] Z.-B. Zhou, D. Zhao, L. Shu, H.-C. Chao, Efficient Multi-Attribute Query Processing in Heterogeneous Wireless Sensor Networks, *Journal of Internet Technology*, Vol. 15, No. 5, pp. 699-712, September, 2014.

[18] E. Jenelius, L. G. Mattsson. Road Network Vulnerability Analysis: Conceptualization, Implementation and Application, *Computers, Environment and Urban Systems*, Vol. 49, pp. 136-147, January, 2015.

[19] M. Ranganathan, A. Acharya, J. Saltz, Distributed Resource Monitors for Mobile Objects, *Proceedings of the Fifth International Workshop on Object-Orientation in Operating Systems,* Seattle, WA, 1996, pp. 19-23.

[20] T. C. Du, E. Y. Li, A.-P. Chang, Mobile Agents in Distributed Network Management, *Communications of the Association for Computing Machinery*, Vol. 46, No. 7, pp. 127-132, July, 2003.

[21] H. Yang, M. G. H. Bell, Models and Algorithms for Road Network Design: A Review and Some New Developments, *Transport Reviews*, Vol. 18, No. 3, pp. 257-278, March, 1998.

[22] L. J. LeBlanc, E. K. Morlok, W. P. Pierskalla, An Efficient Approach to Solving the Road Network Equilibrium Traffic Assignment Problem, *Transportation Research*, Vol. 9, No. 5,

pp. 309-318, October, 1975.

[23] J. Feng, J. Lu, Y. Zhu, N. Mukai, T. Watanabe, Indexing of Moving Objects on Road Network Using Composite Structure, *Knowledge-Based Intelligent Information and Engineering Systems*, Vietri sul Mare, Italy, 2007, pp. 1097-1104.

[24] J. Feng, T. Watanabe, *Index and Query Methods in Road Networks*, Springer, 2015.

[25] C. Bettstetter, G. Resta, P. Santi, The Node Distribution of the Random Waypoint Mobility Model for Wireless Ad Hoc Networks, *Institute of Electrical and Electronics Engineers Transactions on Mobile Computing*, Vol. 2, No. 3, pp. 257-269, September, 2003.

[26] C. Bettstetter, H. Hartenstein, X. Perez-Costa, Stochastic Properties of the Random Waypoint Mobility Model, *Wireless Networks*, Vol. 10, No. 5, pp. 555-567, September, 2004.

[27] Google Maps, https://maps.google.com.tw, October, 2015.

[28] K.-C. Lai, *Improving Zone-disjoint Multipath Routing Algorithm for Mobile Ad-hoc Networks*, Master's Thesis, Tatung University, Taipei, Taiwan, 2011.

Biographies

Chow-Sing Lin received the Ph.D. degree in Computer Engineering from the University of Central Florida, Florida, USA, in 2000. He is currently a Professor in the Department of Computer Science and Information Engineering, National University of Tainan, Taiwan. His research interests include media stream delivery, mobile computing, and media content analyses.

Ci-Ruei Jiang is currently a graduate student in the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan. His research interests include mobile computing, machine learning, and wireless network.