Int. J. Communications, Network and System Sciences, **2011, 4, 313-322**
doi:10.4236/ijcns.2011.45036 Published Online May 2011 (http://www.SciRP.org/journal/ijcns)

A Dynamic Interval Based Circular Safe Region Algorithm for Continuous Queries on Moving Objects

Shengsheng Wang[1], Chen Zhang[2]
[1]*College of Computer Science and Technology*, *Jilin University*, *Changchun*, *China*
[2]*Department of Computer Information Systems*, *Bryant University*, *Smithfield*, *Rhode Island, USA*
E-mail: *wss@jlu.edu.cn*, *czhang@bryant.edu*
Received February 19, 2011; *revised April* 3, 2011; *accepted April* 16, 2011

Abstract

Moving object database (MOD) engine is the foundation of Location-Based Service (LBS) information systems. Continuous queries are important in spatial-temporal reasoning of a MOD. The communication costs were the bottleneck for improving query efficiency until the rectangular safe region algorithm partly solved this problem. However, this algorithm can be further improved, as we demonstrate with the dynamic interval based continuous queries algorithm on moving objects. Two components, circular safe region and dynamic intervals were adopted by our algorithm. Theoretical proof and experimental results show that our algorithm substantially outperforms the traditional periodic monitoring and the rectangular safe region algorithm in terms of monitoring accuracy, reducing communication costs and server CPU time. Moreover, in our algorithm, the mobile terminals do not need to have any computational ability.

Keywords: Location Based Service, Moving Object Database, Continuous Spatial Query

1. Introduction

The development of technology has made it possible to track moving objects such as vehicles, aircrafts, vessels, wildlife, and human objects such as firefighters in a fire field. Technologies such as global positioning system (GPS), radio-frequency identification (RFID), cellular wireless networks (such as commercial cellular phone networks) and even triangulated wireless fidelity (Wi-Fi) networks can all provide location information in real-time, although at different precisions with different effective ranges.

Two major trends can be identified to manage the large amount of location and property information that varies with time: moving object databases (MOD) and data stream technology (DST). The first approach implies extending traditional database techniques with models and index structures suitable to track the locations of the moving objects efficiently. The second approach focuses on the processing of continuous location updates as they arrive. The boundary between these two approaches is not always clear in relation to the topic of this survey: Both propose alternatives to classical database techniques, which are not considered appropriate to manage the continuously changing locations of the moving objects [1]. Our research will focus on the MOD approach but can be easily modified to adapt to the DST approach as well since our Dynamic Interval Based Circular Safe Region (DIBCSR) algorithm requires the minimum frequency of location updates which can be provided by both approaches.

MOD is a system that performs storage management and query analysis on time-variable spatial information of moving objects [2-4] which combines multiple disciplines and research areas including geographical information systems (GIS), spatial databases, spatial-temporal databases, computer graphics, computational geometry, artificial intelligence and mobile computing.

Application of MOD requires the optimal efficiency of the queries which can only be provided by continuous spatial-temporal queries. A regular spatial-temporal query only returns a single result set. In contrast, a continuous spatial-temporal query returns result sets continuously from the registration to the cancellation of the query, which is called the effective period of the query. Even if the query conditions remain unchanged during the effective period, the query result must be updated continuously due to the continuous movement of the queried objects. Here are two examples of continuous spatial-temporal queries which provide commonly used LBS

such as range query or the k-nearest neighbor (kNN) query:

1) List all vehicles that appear in region R in the next 10 minutes.

2) Continuously mark the ten closest vehicles to gas station number five.

These types of queries are not commonly supported by traditional relational database engines. In order to facilitate these continuous spatial-temporal queries, a MOD engine must implement the query processing and ideally, with optimal performance at a low cost.

2. Related Works

Performance of the dynamic updates of the query result set during the effective period is the main research topic of MOD and spatial-temporal reasoning. In order to perform continuous query optimization in a distributed system, not only the query cost must be minimized, the communication cost for updating location information of the terminal devices must also be minimized. However, most of the previous works on continuous queries have focused on reducing the query cost and has ignored the communication cost [5-9], in which the occasions for reporting location information are determined by the terminal device at fixed intervals or when the object's location (constant distance interval) experiences a significant change. This class of uniform time/distance interval strategies has the following weaknesses:

1) The location updates are not adaptive to queries. When queries are scarce or there are no queries at all, a large amount of communication bandwidth and battery power of the terminal device may be wasted on the updates.

2) Low efficiency of queries could cause inconsistency with reality. In the periodical update strategy, improving the consistency of the query result with reality relies on the location update frequency increase. This means higher communication costs and may even make the improvement impossible because of the bandwidth and network delay limitation.

3) Unbalanced workload is applied on the server. In order to improve the reality consistency, the server must update large amounts of location information constantly and recalculate all the queries. An overloaded server usually means low responsiveness and poor reliability.

Hu, *et al.* [10] proposed a continuous query update strategy based on a rectangular safe region (RSR) method which can alleviate the previous three problems. However, with analysis and experiments, we found that this strategy requires considerable computation power on terminal devices. This performance bottleneck may become more significant with a larger query load.

We propose an improved dynamic interval based circular safe region (DIBCSR) strategy to replace the RSR strategy. In our strategy, the location update information is updated dynamically by the server and the terminal devices do not need to download the safe region information. Experiments show that our strategy eliminates the computation requirements of the terminal devices with equal or better system performance than RSR strategy.

Moreover, most of the previous studies only support one specific query type, such as either range queries kNN queries but not both. Our proposed DIBCSR algorithm supports both range queries and kNN queries.

3. Safe Region Based Location Updates

Figure 1 demonstrates the infrastructure of a moving object query system. The kernel of the system is the control center (the main server of the system) in the center of the figure which runs the MOD engine, collects location information, handles continues queries and provides query results to the application servers to the right of the figure. Therefore, the major computation workload is applied to the main server/control center of a MOD system. For simplicity, we refer to the main server/control center as server in this paper.

Terminal devices, which are the monitored moving objects, obtain their own location information from the GPS system and transmit it to the server via a wireless communication network. The whole system's timeliness and efficiency is affected by the wireless communication bandwidth. The location information updates are often the bottleneck because of the limited wireless bandwidth and the high sampling rate in the traditional uniform time/distance interval strategies.

The idea behind the rectangular safe region (RSR) algorithm [10] is to define a rectangular safe region for every object according to the registered query and the latest location obtained. As long as the object's motions do not exceed its safe region, all the query result sets of the object remain unchanged (**Figure 2**). The terminal device is informed of the safe region assignments dynamically. Hence when a terminal device finds that it has exceeded the safe region, it will report its new location information. E.g., when an object a in **Figure 2** has moved out of its safe region of S_a to location a', it will report its new location to the server which will recalculate the results of a continuous k-nearest-neighbor (kNN) query $Q1$ and a range query $Q2$.

Through analysis and experiments, we found that although the RSR algorithm is effective, it has the following weaknesses which can be improved:

1) RSR requires that the terminal device has memory

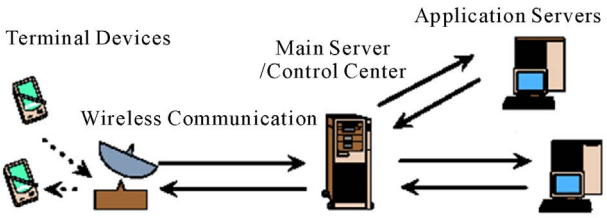

Figure 1. Infrastructure of the MOD system.

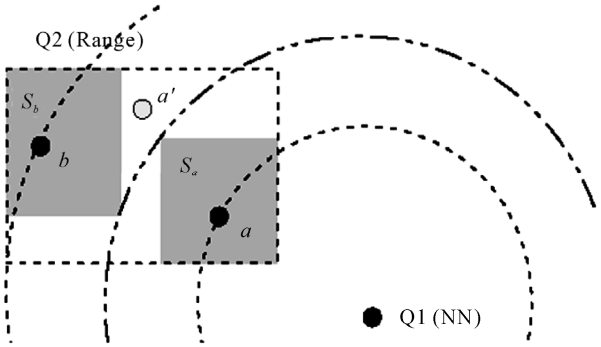

Figure 2. Rectangular safe region.

to store its current safe region and computing power to determine whether it has exceeded the safe region. However, in practice, many low-cost terminal devices (e.g. a GPS dog collar) do not have memory and computing power in addition to GPS satellites communication.

2) In RSR, data communication is bidirectional. The terminal devices not only need to upload location information to the server, but also need to download safe region information from the server. When the query frequency increases, the frequency of safe region download to the terminal devices increases. When the query frequency is high enough, the communication cost may be even worse than the uniform time interval (UTI) strategy. We have a detailed analysis of this problem in Section 5.2.

3) Computations involved in the RSR strategy are complicated, especially for kNN queries.

On the observation of these problems, we propose a new continuous query algorithm. We define the safe region of object *o* (referred to as *o.sr*) as a circle with the center at the location of the object and the radius of *o.r* (**Figure 3**). Assume the maximum speed of the object is *o.maxspd*, then the continuous query result of query *q* will not be affected within the time interval of *o.r/o.maxspd*. Hence the server can issue a location report query to the terminal device at the time of (*o.r/o.maxspd* - δ) where δ is the sum of communication and computation delays.

In comparison, the advantages of our DIBCSR strategy are:

1) The terminal device does not need to have any computing power. The only task of the terminal device is to report its location upon the request of the server. This is determined by our main algorithm which is given in Section 4.1. Moreover, the location update sampling requests are distributed by the server. Therefore, when the sampling strategy needs updating, such as when safe regions are reassigned because of objects' movements, only the server is affected and the communication cost will not be affected.

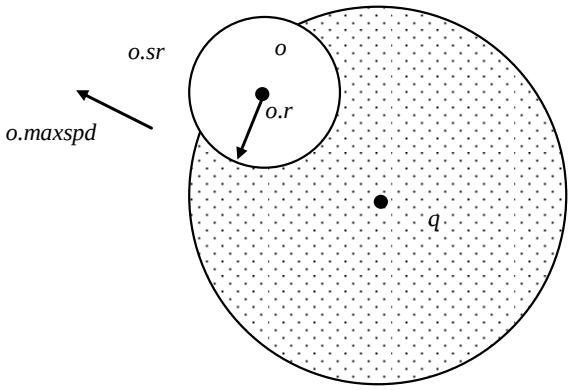

Figure 3. Location update of a moving object with circular safe region under continuous query.

2) The algorithm determining the assignment of a circular safe region is simpler than a rectangular one. Therefore the computation is reduced for safe region assignments. We provide the detailed safe region assignment algorithms for continuous range queries and continuous kNN queries in Sections 4.2 and 4.3 respectively.

3) The updates of safe regions have been minimized with the selection of circular shape safe regions and therefore the communication cost is minimized. We provide mathematical analysis in Section 5.1.

4) The communication cost is reduced in comparison with the RSR strategy. Detailed analysis of this feature is provided in Section 5.2.

5) Computations involved in the RSR strategy are complicated, especially for kNN queries. In contrast, computations are much more concise in our DIBCSR strategy.

4. Dynamic Interval Based Location Updates

We use a C++/Java style pseudo code syntax, including comment syntax of double slash, to represent the algorithms in a more concisely and precisely. Properties of the moving object *o* and continuous query *q* are explained in **Table 1** and **Table 2**.

A separate process will be responsible for determination of the objects that are due for reporting new locations and sending the requests. The main algorithm

Table 1. Properties of a moving object.

q.region	Query region of a range query
q.result	Query result set
q.effUTI	the query effective period
o.circle.p	the center of the circular query region
o.circle.r	the radius of the circular query region

Table 2. Properties of a continuous query.

o.p	Last reported object location
o.r	Radius of the object safe region
o.sr	Object safe region
o.maxspd	Maximum speed of the moving object
o.upt	Next location update time of the moving object

which runs on the server is as following:

***Algorithm* 1. *Main algorithm for continuous query processing*:**

```
//OList is the object list, QList is the query list
while (received query q and
current time t within q.effUTI)
do
{
if (q is newly registered) then
{
//new a query q for processing, either //range or kNN query
NewQuery(q);
}
if (q is cancelled) then
   remove q from QList;
if (q is location update of object o) then
{
//update safe region of object o and //related query result sets
UpdateSR(QList, o);
   o.upt = t + o.r/o.maxspd - delay;
}
}

UpdateSR(QList, o)
{
//range query location updates processing UpdateSRA(QList, o);
//kNN query location updates processing UpdateRSR(QList, o);
}
```

4.1. The Main Algorithm for Continuous Query Processing

Both continuous range query and continuous kNN query are supported by our strategy. A continuous range query is one that returns all the objects in *q.region* within the query effective period where query region can be either rectangular or circular. A continuous kNN query is one that returns the closest *k* objects to the query location. An ordered kNN query requires the results to be returned in increasing order and an unordered kNN query does not request the results to be in order. An ordered kNN query is what we will consider in this paper and which is more complicated than an unordered kNN query. These two different types of continuous queries require different new query processing and location update processing algorithms which we present in different sections as follows.

4.2. Continuous Range Queries

The query processing and location update algorithm for continuous range queries are as follows:

Algorithm* 2. *New range query processing

```
NewQuery(q)
//q is a range query
{
for (∀o ∈ OList) do
{
qResult = Req(q, o.sr);        //query result
if (qResult = "Y" ) then
q.result= q.result ∪ {o};
// added to the result set of q
// if the query result of is undecided
  if (qResult = "U" ) then
{
Query location of object o;
//object o must report immediately
//recalculation is then performed.
UpdateSR(QList, o);
}
}
}
```

In a range query, the Req function in Algorithm 2 which is the query result of safe region is defined as

$$\mathrm{Req}(q, o.sr) = \begin{cases} Y & \text{if } RCC5(q.r, o.sr) = PPI \\ N & \text{if } RCC5(q.r, o.sr) = DC \\ U & \text{otherwise} \end{cases} \quad (1)$$

A return value of "*Y*" indicates that the safe region is inside the query region and therefore object *o* is within the result set. A return value of "*N*" indicates that the safe region is outside of the query region and therefore the object *o* is not included in the result set. A return value of "*U*" indicates that the safe region intersects with the query region. The result is therefore undecided. Hence the precise location of the object needs to be ob-

tained in order to recalculate. The region connection calculus (*RCC*) serves for qualitative spatial representation and reasoning and *RCC*-5 is a widely used binary relationship model in automated spatial reasoning [11] with five binary relationships {DC, PO, PP, EQ, PPI} (discreteness, proper overlap, proper part, equivalence, proper inclusion) demonstrated in **Figure 4**. The function *RCC*5(X, Y) returns the *RCC*-5 relationship of X and Y for topology analysis.

Algorithm* 3. *Range query update

//The functionality is to update the range query //result set and check/update the object safe //region by invoking the SafeRegion sub //function.

UpdateSRA(*QList*, *o*)
{
o.r = ∞;
for (∀*q* ∈ *QList* and *q* is a range query) do
{
// If safe region function returns *r*>0
// then *o* is within the result set of *q*
if SafeRegion(*q*, *o*, *r*) then{
q.result = *q.result* ∪ {*o*};
return *true*;
}
}
}

Algorithm* 4. *The calculation of safe region for range query

The purpose of this algorithm is to calculate the safe region radius and decide the query result set. The safe region radius *r* is returned for the object *o* under query *q*. The Boolean return value of *true* or *false* indicates whether object *o* is within the query result set.

SafeRegion(*q*, *o*, & *r*)
{
if (*q* is a rectangular range query) then
{
//Three circumstances exist,
//as demonstrated in **Figure 5**.
//A: *o.p* is inside query region I
//B: *o.p* is inside query region II
//C: *o.p* is inside query region III
if (*o.p* is inside query region I) then
{
r = distance from *o.p* to the closest edge of the rectangular query region;
return *true*;
}
else if (*o.p* is inside query region II) then
r = distance from *o.p* to the closest edge of the rectangular query region;
else // *o.p* is inside query region III
r = distance from *o.p* to the closest vertex of the rectangular query region;
return *false*;
}
else if (*q* is a circular range query) then
{
//Two circumstances exist,
//as demonstrated in **Figure 6**.
//A: object *o* is inside the
//circular query region
//B: object *o* is outside of the
//circular query region
doq= *dist*(*o.p*, *q.circle.p*);
//*dist*(*a*, *b*) represents the distance
//between point *a* and point *b*. *doq* is the //distance between object *o* and query *q*.
r = ABS(*doq* - *q.circle.r*);
if (*doq* <= *q.circle.r*) then
return *true*;
else
return *false*;
}
}

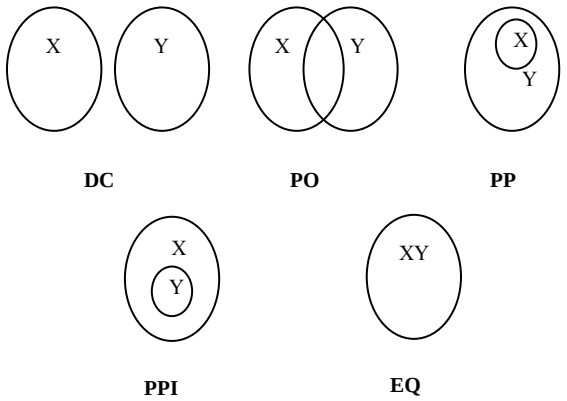

Figure 4. Binary spatial relationships in RCC-5.

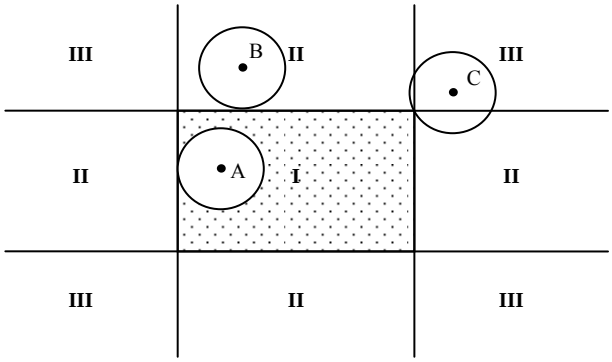

Figure 5. Rectangular range query safe region.

Theorem 1: When the motion of the object in Algorithm 4 does not exceed the safe region, the result set of

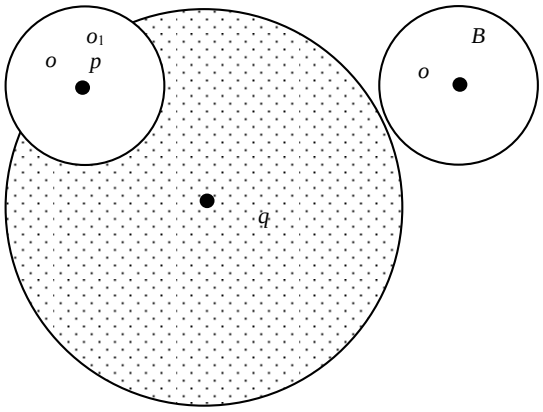

Figure 6. Circular range query safe region.

the continuous range query does not change.

Proof: *Apparently, under the circumstance, o.sr does not intersect with q.r. Hence object o's motion inside o.sr will not affect the query result set.*

4.3. Continuous kNN Query

Algorithm* 5.** ***An ordered kNN query processing

NewQuery(*q*)
//*q* is an ordered kNN query
{
1) Decide the object list *OList* near query location *q.p* based on the spatial-temporal index of moving objects;
/*e.g. objects within neighboring rectangles can be selected in an R*-tree indexed system. This step reduces the object set that is processed to reduce the following computation.*/
2) Perform sorting to the objects in *OList* by dist (*o.p*, *q.circle.p*) ascending, the nearest k + 1objects are saved in *q.result*;
/*Quick Sort algorithm is applied and the Compare function is given below. The reason why we save the (k +1)th object is for the calculation of the safe region.*/
3) Update the safe regions for all objects;
}

Algorithm* 6.** ***Distance comparison algorithm for objects

For simplicity, we use *q* to represent *q.circle.p* and object names *o*1, *o*2 to represent the object location *o*1.*p* and *o*2.*p* in this section.

// Function returns –1 when *o*1 is nearer than *o*2; // returns 1 when *o*1 is farther than *o*2.

// Since all calculations are floating-point, we do // not consider the equal scenario.

Compare (*q*, *o*1, *o*2)
{
if (*dist*(*q*, *o*1) + *o*1.*r* < *dist*(*q*, *o*2) – *o*2.*r*) then
return –1;
if (*dist*(*q*, *o*1) – *o*1.*r* > *dist*(*q*, *o*2) + *o*2.*r*) then
return 1;
Query locations of *o*1, *o*2;
//Distances of safe regions to *q* overlaps, //query precise locations for further //comparison.
*o*1.*r* = 0;
*o*2.*r* = 0;
if (*dist*(*q*, *o*1) < *dist*(*q*, *o*2)) then
return –1;
else
return 1;
}

4.4. Circular Safe Region Calculation and Updates for Continuous kNN Queries

Following is the formula to update the safe region radius for the i^{th} object in the object set ascending sorted by distance to ordered kNN query *q* in Algorithm 5. **Figure 7** shows an example of safe regions assignment in such an ordered kNN query *q*. The first object in the result set is *q* and the extra object o_{k+1} is kept for calculation of safe region radius of o_k.

$$o_i.r = \begin{cases} \text{if } 0 < i \le k, \text{ in the result set, then} \\ \min\left\{o_i.r, \dfrac{dist(o_i, o_{i-1})}{2}, \dfrac{dist(o_i, o_{i+1})}{2}\right\} \\ \text{if } i > k, \text{ out of the result set, then} \\ \min\left\{o_i.r, dist(o_i, q) - quar(q)\right\} \end{cases} \quad (2)$$

where *quar*(*q*) is the radius of the quarantine region for query *q* which surround and only surround the safe regions of all objects in the result set. Therefore, *quar*(*q*) = *dist*(o_k, *q*) + o_k. **Figure 8** shows how such a quarantine region is assigned for such an ordered kNN query *q*. Hence we have the following property: either in or out of the kNN query result set (inside or outside the quarantine region), none of the safe regions of objects overlaps with each other. Therefore, when all the objects are moving inside their own safe regions, the result set and its order are not affected.

4.5. Location Update Processing for Continuous kNN Queries

For continuous kNN query, when object location is updated, one of the four following scenarios will happen. The detailed update algorithm is given in Algorithm 7:

1) Original location was inside the quarantine region and new location is also inside the quarantine region: order adjustment in the result set is necessary.

2) Original location was outside the quarantine region but new location is inside the quarantine region: a new

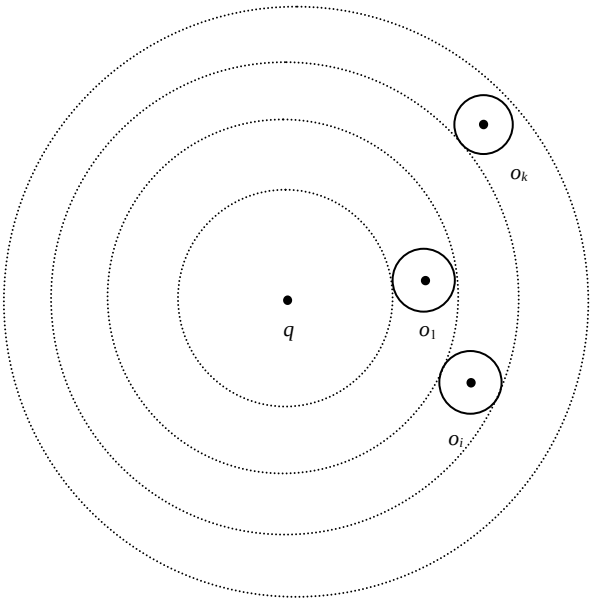

Figure 7. Safe regions assignment in an ordered kNN query.

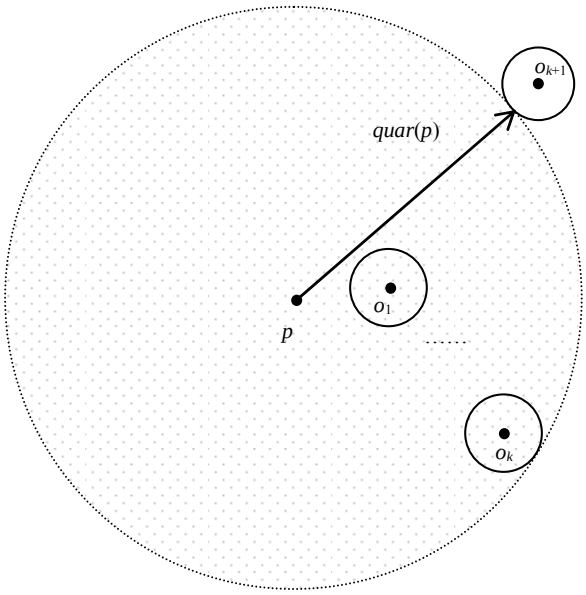

Figure 8. Quarantine region assignment for an ordered kNN query.

query is necessary to recalculate the complete result set.

3) Original location was inside the quarantine region but new location is outside the quarantine region: same as 2).

Both original location and new location are outside the quarantine region: only need to update the object's safe region.

Algorithm 7. Location update processing for kNN query

//Purpose is to update safe region and result set

UpdateRSR(*QList*, *o*)
{
for ($\forall q \in QList$ and q is a kNN query) do
{
if (($o \notin q.result$ and $dist(o, q) \le quar(q)$) or ($o \in q.result$ and $dist(o,q) > quar(q)$)) then
{
Execute Algorithm 5;
//Processed as a new query
continue;
}
else if ($o \in q.result$
and $dist(o, q) \le quar(q)$) then
{
//Original result set of q is $\{o_i | i = 1,\cdots, k\}$, //if object o's original index is i and //index after the new sorting by $dist(q, o)$ //is i'
Execute Algorithm 5 within the range of

$$\begin{cases} i-1, i, \cdots, i', i'+1, & \text{when } i' \ge i \\ i'-1, i', \cdots, i, i+1, & \text{when } i \ge i' \end{cases}$$

Use result set from Algorithm 5 to replace the respective subset in the original result set;
continue;
}
else //$o \notin q.result$ and $dist(o, q) > quar(q)$
{
Adjust $o.r$ following the Req function given by Equation (1);
continue;
}
}
}

5. Analysis and Experiment

5.1. Analysis of the Safe Region Shape

As previously mentioned in Section 3, one reason for selecting the circular safe region shape is to minimize the updates of safe regions and therefore the associated communication cost. We further provide mathematical analysis here.

In a safe region based strategy, the communication cost of a location update is inversely proportional to the minimum location update time. This is because the object's motion direction is generally unpredictable. Therefore its probability of leaving the safe region through any portion of the border is equal. Assume *SR* to be the safe region, p to be the last reported location. If in the direction of θ, the distance from p to the edge of the safe region is $k(\theta)$ (**Figure 9**), then the minimum location update time is

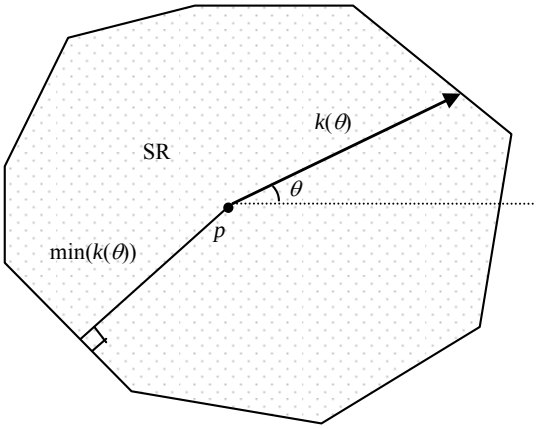

Figure 9. Shape of the safe region.

$$1/cost_{communication} \propto time_{update} = \min\left(k\left(\theta\right)\right)/o.maxspd \quad (3)$$

Assuming *o.maxspd* is a property of the object that we cannot control, our goal is to maximize the min($k(\theta)$) in order to maximize the t_{update}. When a specific shape of safe region is selected, increasing the area of the safe region is obviously going to increase min($k(\theta)$). However, in a range query or a kNN query, the largest area is eventually bounded by the objects' distribution and the size of the query region. Therefore, if we assume the area of the safe region is determined, then the maximized average distance from p to the edge of the safe region in all directions, min($k(\theta)$), will come with the isotropic safe region – a circle.

5.2. Analysis of the Safe Region's Communication Cost

In the RSR strategy [10], data communication is bi-directional: the terminal devices upload location information and download the safe region information. However, the authors only discussed the communication cost of location information upload, which is not precise. We take the bi-directional data communication into consideration in the following discussion and then compare the communication cost with our DIBSCR strategy.

In the RSR strategy, communication in the down-link direction from the server to the terminal device transmits information of the rectangular safe region, each determined by two points or four coordinates. Communication in the up-link direction from the terminal device to the server transmits a location update, each includes one point or two coordinates. In newer wireless networks such as Wi-Fi, Wi-Max or 3G, data is transmitted in data frames (called synchronous transmissions mode). Since the data amount to be transmitted/received by the terminal device every time is quite small which can easily fit into a single frame, the location update rate is the main factor affecting the communication cost. This location update rate is constant for UTI strategy while variable for safe region based strategies. Assume in the RSR strategy, the safe region update rate is s which depends on the query rate of q. We therefore represent it using the function of $s(q)$ which increases with the query rate of q. And assume the location update rate is u and the communication delay is d. The total communication cost is $(s(q) + u)\cdot d$ which increases with the query rate. Hence when the query rate is high, the RSR strategy can be even worse than the UTI which makes it not feasible.

In contrast, in our DIBSCR strategy, the terminal device does not need to download safe region information which only leaves the location update term of $u\cdot d$. Moreover, the location update rate r is most equal to the constant rate in the UTI strategy because it is based on the dynamic time interval which is greater than or equal to a preset value. We further provide the estimated location update rate r in DIBSCR as following:

The basic estimate of the location update interval is derived from Equation (3):

$$1/u \propto t_{update} = o.r/o.maxspd \quad (4)$$

where t_{update} is the minimum amount of time the object may exceed the circular safe region and therefore requests a location update. The estimate of the location update rate u is therefore relying on the estimate of the object's maximum speed *o.maxspd*.

The estimate of *o.maxspd* can be either fixed (for instance the object types of pedestrian, motor vehicle or high-speed train) or it could also be further refined by prediction from the object's historical locations. This could reduce the communication cost when the object is temporarily immobile (such as when the pedestrian stopped by a coffee shop or when the motor vehicle is parked). Certainly in any prediction based speed estimate, we need to be on the conservative side and control the computation cost although there are outstanding prediction methods such as Back Propagation Networks (BPN). For simplicity, we do not want to include consideration of a missing rate. One possible conservative estimate is

$$o.maxspd = \min\left(fixed\_max\_speed, \right.$$
$$\left. v_{last} + max\_acceleration * \left(t_{current} - t_{last}\right)\right)$$

where

fixed_max_speed: the maximum possible speed for an object type

v_{last}: calculated speed at the last location report time

max_acceleration: the maximum possible acceleration of an object type

$t_{current}$: the current system time

t_{last}: the last location report time.

Either the *o.maxspd* is fixed or further bounded by a prediction, it is tightly bounded and hence the minimum amount of time the object may exceed the circular safe region and the maximum location updated rate is tightly bounded and independent of the query rate.

5.3. Experimental Analysis of the System Performance

In order to further evaluate our strategy, we constructed a simulation system to evaluate the UTI strategy, RSR strategy and our DIBCSR strategy. In our simulation system, object *o*'s motion direction and speed are randomly generated. The object's speed should not exceed a fixed maximum speed of *o.maxspd*. In UTI simulation, we have two location update intervals of 0.1s and 1s, referred to as UTI(0.1) and UTI(1) respectively in the simulation results.

In our experimental analysis through simulation, three comparison criteria are applied: 1) precision, 2) communication cost and 3) server workload. We analyze the simulation results separately in the following section.

1) Precision

The precision of a continuous query result is defined as: at time *t*, the system query result is RESULT(*t*); the actual object set that satisfies the query condition is TRESULT(*t*), standing for the true result. In order to use a higher value to represent higher precision, we define the *equal*(*x*, *y*) function to return 1 when $x = y$ and 0 when $x \neq y$. In the time interval of [a, b], we average the equality between the returned value and the true value, and define precision as

$$\frac{1}{b-a}\int_b^a equal\left(RESULT(t), TRESULT(t)\right)\mathrm{d}t$$

The precision is obviously affected by the communication delay since it causes the difference between the actual location and the reported location. The result is represented in **Figure 10** and the precision of DIBCSR and SBR are approximately the same and both are better than UTI.

The simulation results shown in **Figure 11** confirm our analysis in Section 5.2 that the performance of DIBCSR is significantly better (lower communication cost) than RSR when considering bi-directional communication cost. And at high query rate, communication cost of RSR can be even worse than UTI with a larger time interval (lower sampling rate).

2) Communication cost

3) System scalability

Figure 12 shows the comparison of server workload for different strategies under the query rate increase. Since the workload is balanced better, both DIBCSR and

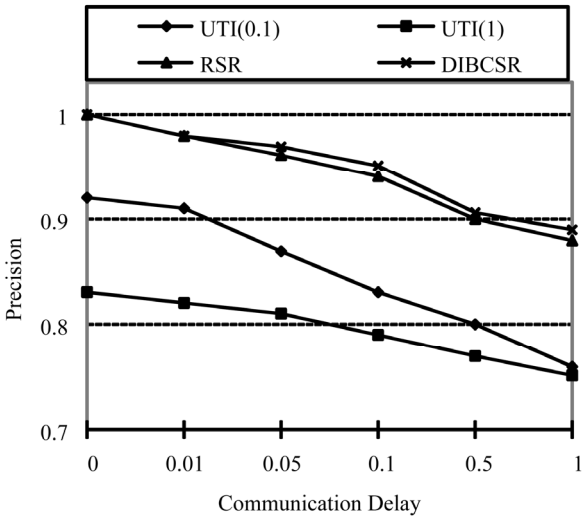

Figure 10. Comparison of precision.

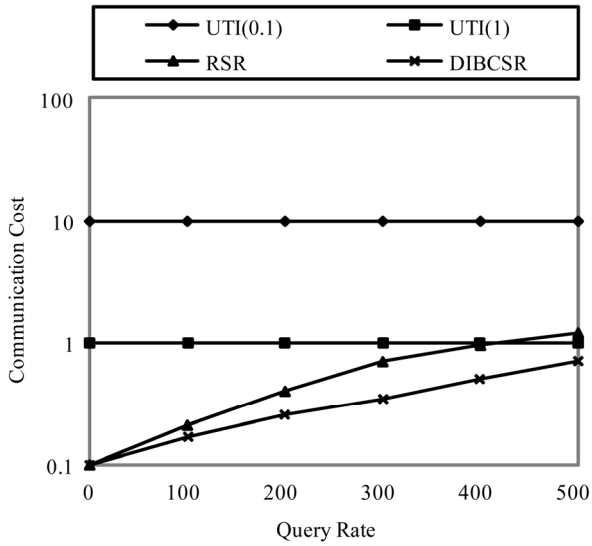

Figure 11. Comparison of communication cost.

RSR apply less workload on the server than UTI. Because we further simplified the safe region computation by applying circular safe region, DIBCSR applies even less workload than RSR on the server. This advantage is more significant under the query rate increase. This low server workload feature of DIBCSR helps to improve the system scalability.

6. Conclusions

This paper analyzes the weaknesses of the RSR strategy and proposes a DIBCSR strategy to replace the RSR strategy for continuous queries in MOD. Theoretical analysis and simulation experiment both show that the new strategy has multiple advantages. Firstly, the new

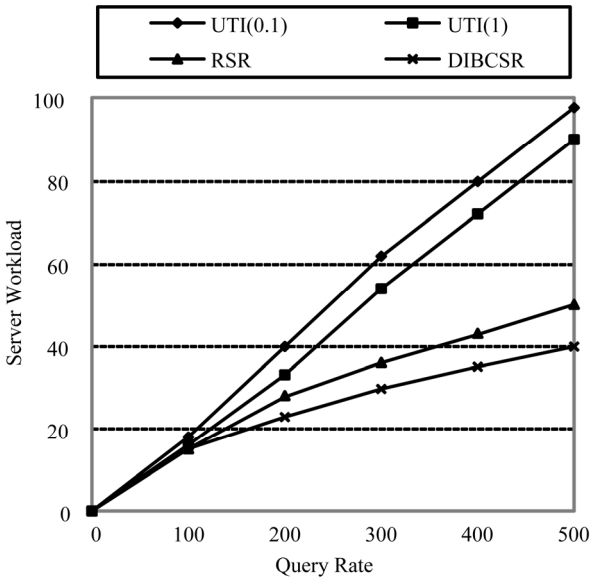

Figure 12. Comparison of server workload.

strategy does not require computation over the terminal devices. Therefore, cost of the terminal devices is reduced under the precondition of equal or better system performance. Secondly, terminal devices do not need to download the safe region information from the server which reduces the communication cost effectively. Finally, computation is simplified by applying circular safe regions. Hence the server workload is reduced which improves the system scalability. Possible future works of the research include implementation of the strategy in an applied MOD engine for a information system providing LBS to public transportation, taxis and private vehicle devices or pedestrians with hand-held mobile devices. Application of our strategy potentially provides real-time range queries and kNN queries to support LBS at a low cost with a high performance in addition to system design and implementation ease and flexibility.

7. References

[1] S. Ilarri, E. Mena and A. Illarramendi, "Location-Dependent Query Processing: Where We Are and Where We Are Heading," *ACM Computing Surveys*, Vol. 42, No. 3, 2010, pp. 1-79. doi:10.1145/1670679.1670682

[2] H. D. Chon, D. Agrawal and A. El Abbadi, "Storage and Retrieval of Moving Objects," *Proceedings of the 2nd International Conference on Mobile Data Management*, Hong Kong, 8-10 January 2001, pp. 173-184.

[3] H. D. Chon, D. Agrawal and A. El Abbadi, "FATES: Finding a Time Dependent Shortest Path," *Proceedings of the 4th International Conference on Mobile Data Management*, Melbourne, 21-24 January 2003, pp. 165-180.

[4] Y. Chen, F. Rao, X. Yu and D. Liu, "CAMEL: A Moving Object Database Approach for Intelligent Location Aware Services," *Lecture Notes in Computer Science*, Vol. 2574, 2003, pp. 331-334. doi:10.1007/3-540-36389-0_23

[5] C. S. Jensen, D. Lin and B. C. Ooi, "Query and Update Efficient B^+-Tree Based Indexing of Moving Objects," *Proceedings of the 30th International Conference on Very Large Data Bases*, Toronto, 29 August - 3 September 2004, pp. 768-779.

[6] M. F. Mokbel, X. Xiong and W. G. Aref, "SINA: Scalable Incremental Processing of Continuous Queries in Spatio-Temporal Databases," *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, Paris, 13-18 June 2004.

[7] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref and S. E. Hambrusch, "Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects," *IEEE Transactions on Computers*, Vol. 51, No. 10, 2002, pp. 1124-1140. doi:10.1109/TC.2002.1039840

[8] Y. Tao, D. Papadias and Q. Shen, "Continuous Nearest Neighbor Search," *28th International Conference on Very Large Data Bases*, Hong Kong, 20-23 August 2002. doi:10.1016/B978-155860869-6/50033-0

[9] X. Yu, K. Q. Pu and N. Koudas, "Monitoring K-Nearest Neighbor Queries over Moving Objects," *21st International Conference on Data Engineering*, Tokyo, 5-8 April 2005, pp. 631-642

[10] H. Hu, J. Xu and D.L. Lee, "A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects," *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, Baltimore, 13-16 June 2005. doi:10.1145/1066157.1066212

[11] D. A. Randell, A.G. Cohn and Z. Cui, "Computing Transitivity Tables: A Challenge for Automated Theorem Provers," *Lecture Notes in Computer Science*, Vol. 607, 1992, pp. 786-790.

