2003 International Symposium on Advanced Intelligent Systems
September 25-28, 2003, Jeju, Korea

The Method of Continuous Nearest Neighbor Search on Trajectory of Moving Objects

*BoYoon Choi, *SangHo Kim, **KwangWoo Nam, *KeunHo Ryu
* Database Laboratory, Chungbuk National University, CheongJu Chungbuk, Korea
** Electronics and Telecommunications Research Institute, Daejun Chungnam, Korea
e-mail : *{ bychoi, shkim, khryu } @ dblab.cbu.ac.kr , ** kwnam@etri.re.kr

Abstract - **When user wants to find objects which have the nearest position from him, we use the nearest neighbor (NN) query. The GIS applications, such as navigation system and traffic control system, require processing of NN query for moving objects (MOs). MOs have trajectory with changing their position over time. Therefore, we should be able to find NN object continuously changing over the whole query time when process NN query for MOs, as well as moving nearby on trajectory of query. However, none of previous works consider trajectory information between objects. Therefore, we propose a method of continuous NN query for trajectory of MOs. We call this CTNN (continuous trajectory NN) technique. It can find constantly valid NN object on the whole query time by considering of trajectory information.**

I. INTRODUCTION

Recently, with the development of mobile/wireless computing technologies or GIS/GPS techniques, works on location-based services have been briskly progressing. To offer these services it is necessary to execute various types of query efficiently, such as range query, NN query, etc. One of the most important queries among them is NN query. Many works on NN query have been progressed up to now.

Location-based services are often applied to MOs. And then MOs have trajectory as their position is changed over time. So if we process NN query for MOs with traditional techniques (for example [1]), it evaluate query at once on position of any instant time. Then returned information becomes invalid when they move. So, we need a timely response as well as efficient and scalable execution for query. And we also must consider trajectory information of MOs. That is, we must grasp a state whether query object and data objects are getting near to of far from each other, or where are they going to. However, there are no previous works which consider trajectory information. Hence we propose a novel method of NN query for MOs, CTNN (continuous trajectory NN). Porkaew[2] says NN query on spatiotemporal database has two types. One is combined temporal range query and spatial NN query, the other is reverse. Our approach follows the former case.

The rest of this paper proceeds as follows. Related works and problem descriptions are discussed in Section 2. Section 3 describes basic assumption and data model. Our proposed technique is introduced in Section 4. Finally, we conclude this paper in Section 5.

This work was supported by University IT Research Center Project and KOSEF RRC Project (Cheongju Univ. ICRC) in Korea.

II. PROBLEM DESCRIPTIONS & RELATED WORKS

There are some works apply NN query by the case that both data and query are MOs [3, 4, 5]. Kollios[3] proposes a method using Duality Transform technique, but that cannot be applied on more than two dimensions. Benetis[4] calculates distance between MOs using differentiation at periodical instant time on three dimensions. Still they can't guarantee the selected result at other time. Besides it suffers from the usual drawbacks of sampling. For overcome these problems, Tao[5] proposes the continuous NN (CNN) query. Although it finds time points that NN object get changed on query, only k-NN query processing is possible.

Assuming user wants to find one NN object using CNN, the whole query time is divided into several intervals based on time points that NN object get changed. Then it find NN object at each time interval. But if there are several objects that have same nearest distance from query, it has dimness of what objects should be chosen. If we choose an object which maintains the nearest position on query trajectory until next interval, it is possible to return exact one. So it is important to find object not only continuously but also with considering trajectory information. To get trajectory information, we must grasp whether data objects and query object are getting near to or far from each other, or how fast velocity is, as well as how large the slope is, or to which direction they go. By using this information to select NN object, CTNN techniques find more exact and valid result than others.

III. BASIC ASSUMPTIONS & DATA MODEL

We assume that MO is point object and change their position continuously over time. Information of MOs is stored in database with triple <*id*, (x_i, y_i), *t*> whenever updates (insertion, deletion, changing velocity or direction) are occurred. That is, object *id*, spatial coordinate on 2D, and time value when object be in that position, are stored, where *i* denote *i*th storing information of object *id*.

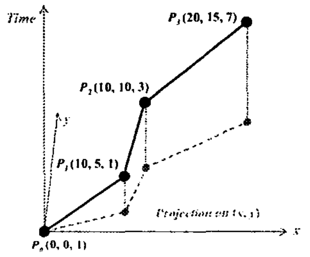


Figure1. Trajectory of Moving Object

In terms of geographic or geometric, the movement path of MO is called trajectory, and that is presented with polyline, i.e. set of separate and sequential segments, as figure 1. Each segment connects consecutive two data points that exist in database. And then each object has a constant velocity on one segment.

We assume the whole trajectory information of data is pre-defined, such as bus, train, and airplane. The spatial coordinate at non-stored time in database can be presumed using function of linear location estimation, as formula (1). This function use the stored coordinate at time t_i and t_{i+1}, then calculate the position at any time point t ($t_i < t < t_{i+1}$).

$$\textit{if}\ (t_i < t < t_{i+1})\ \textit{then}$$
$$f(t) = \left(\frac{x_{i+1} - x_i}{t_{i+1} - t_i}(t - t_i) + x_i \ ,\ \frac{y_{i+1} - y_i}{t_{i+1} - t_i}(t - t_i) + y_i \right) \qquad (1)$$

Additionally, we have following assumptions.

1. If object p is selected as NN object and is changed to another one within 1 minute, then it is ignored and p's valid time interval is contained into former/latter interval. This is too short time interval and such information may be meaningless in real world.
2. If the distance between two objects is less than 1 meter, we regard these objects as meet (or intersect). In real world, if objects have same coordinate at same time, this means collision and these cases hardly occur.

Table 1 shows some notations that used in this paper.

Table1. Notations

$\square a_i$	ith segment of object a ($i \geq 1$, integer)
$\lvert x \rvert$	absolute value
(x_a^t, y_a^t)	spatial coordinate of a at time t
(x_{ai}, y_{ai})	ith stored coordinate value of a
x_{ab} ($\underline{y_{ab}}$)	absolute of difference value between a and b on x (y) axis
Δx (Δy)	movement distance on x (y) axis per unit time interval
$\lvert a\text{-}b \rvert^t = \sqrt{(x_{ab}^t)^2 + (y_{ab}^t)^2}$	spatial distance between a and b at time t
$/p = \Delta y \div \Delta x$	slope of segment p
$+\Delta y, +\Delta x \Rightarrow +/p$; $-\Delta y, -\Delta x \Rightarrow -/p$ $-\Delta y, +\Delta x \Rightarrow -/p$; $+\Delta y, -\Delta x \Rightarrow -/p$	
$\partial p = \sqrt{\Delta x^2 + \Delta y^2}$	displacement of p
$+\Delta y, +\Delta x \Rightarrow +\partial p$; $-\Delta y, -\Delta x \Rightarrow -\partial p$ $-\Delta y, +\Delta x \Rightarrow -\partial p$; $+\Delta y, -\Delta x \Rightarrow -\partial p$	

IV. Continuous Trajectory NN Search

NN query needs to be scalable in terms of the number of total objects and degree of movement of objects. Range query is usually and frequently used in GIS applications as pre-processing tool for reducing the amount of data that other queries, such as NN query, would process. So, CTNN use temporal range query as filtering step. CTNN return the segments of which valid time interval intersect with query interval (temporal filtering), and then find NN object among these on the query trajectory (spatial NN query).

Spatial NN query consist of two calculations. First one is to find *evaluation times* and store *evaluation intervals* in time list *TL*. Evaluation time means a time point when is occurred changing of NN object on query. Evaluation interval is made of consecutive two evaluation time points. Second one is to find valid NN object in each evaluation interval. Information of NN objects is also stored in *TL*. At the end time of query, CTNN return all values in *TL*.

A. Calculation of Displacement

While we find NN object, many objects may have same nearest distance from query. Let these candidates are P. When user wants 1NN query, we must select only one as result among P. In that situation, CTNN compares value of slope and displacement, etc., between $\square P$ and $\square q$. By using this value, we can choose one as NN object p. We call this comparison *calculation of displacement*. The result value of calculation means the first intersection time between two segments. A negative number represents an intersection in past, and a positive one means in future. For example, 2 mean that two segments will intersect after two unit time from current. Therefore, in case of 1NN query we select object that has the smallest calculation value, and in kNN select k objects in order.

Table 2 shows the calculation of displacement between $\square a$ and $\square b$, where h, k, and f is the integer more than 1.

Table2. Calculation of Displacement

1. **When** $/a = /b$ and $\partial a = \partial b$,
   - A. **If** $x_{ab} = y_{ab} = 0$, **Then** $\square a$ and $\square b$ coincide.
   - B. **If** $x_{ab} \neq 0$ and $y_{ab} \neq 0$, **Then** $\square a$ and $\square b$ parallel.
2. **When** $\lvert /a \rvert = 1 / \lvert /b \rvert$ and $\partial a = \partial b$,
   - A. **If** $x_{ab} \neq y_{ab}$ or $x_{ab} = 0$ (or $y_{ab} = 0$) , **Then** $\square a$ and $\square b$ not intersect
   - B. **When** $\lvert /a \rvert = h \div (h+k)$ and $\lvert /b \rvert = (h+k) \div h$, **If** $x_{ab} = y_{ab} = k*f$, **Then** $\square a$ and $\square b$ intersect after f times
3. **When** $\lvert /a \rvert = \lvert /b \rvert < 1$ and $/a = -/b$ and $\partial a = -\partial b$,
   - A. **If** $x_{ab} \neq 0$, **Then** $\square a$ and $\square b$ not intersect
   - B. **When** $x_{ab} = 0$ and $\lvert /a , /b \rvert = h \div (h+k)$, **If** $y_{ab} = h*k*2$, **Then** $\square a$ and $\square b$ intersect after k times.
4. **When** the rest of all cases, calculate

$$X = \frac{x_{ab}}{\partial a - \partial b} \ ,\ Y = \frac{y_{ab}}{\partial a - \partial b}$$

(round off the numbers to two decimal places)

   - A. **If** ($/a$ or $/b$) < 1 or ($/a$ and $/b$) < 1 , **Then** intersection time between $\square a$ and $\square b$ is $X+Y$
   - B. **If** ($/a$ and $/b$) > 1 , **Then** intersection time between $\square a$ and $\square b$ is MAX(X, Y)

B. Temporal Filtering Step of CTNN

Given *Query* = <q, (x_{qs}, x_{qe}, y_{qs}, y_{qe}), [t_s, t_e]>, CTNN start the temporal range query, as table 3, where (x_{qs}, x_{qe}, y_{qs}, y_{qe}) denotes coordinate of end points and [t_s, t_e] is the whole query time interval. This may consist of several segments. The result of temporal filtering have two lists, time list *TL* and coordinate list *CL*.

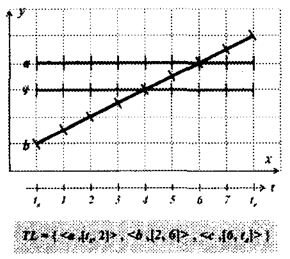
Table3. Temporal Filtering of CTNN

```
1. For all object segments in database, retrieve segments of which
   time intersect with query time interval [$t_s$, $t_e$].
2. For retrieved segments, process the clipping with [$t_s$, $t_e$].
3. For objects that coordinates are unknown at $t_s$ and $t_e$, calculate
   correspond coordinates using formula (1).
4. All endpoints of segments become initial evaluation time points $t_i$
   (except last one), then sequence time intervals [$t_i$, $t_{i+1}$] are stored in
   time list TL. And also, store the coordinate information of them into
   coordinate list CL.
```

C. *Spatial NN query step of CTNN*

After temporal pre-processing, CTNN process spatial NN query using two calculations. First one is to find extra evaluation time, i.e., another time point that NN object is changed (initial time is selected from filtering step and stored in *TL*). Second one is to select NN object within each time interval (this is constituted from extra and initial evaluation time according to the time order).

For simplicity, assume that we apply 1NN query and that objects consist of only one segment, as figure 2. A result of CTNN is *TL* = {<*a*, [t_s, 2]>, <*b*, [2, 6], <*a*, [6, t_e]}. A tuple <*a*, [t_s, 2]> means that object *a* is the NN object within time interval [t_s, 2].



y
a
q
b
x
t
t_s 1 2 3 4 5 6 7 t_e
TL = {<a,[t_s,2]>, <b,[2,6]>, <a,[6,t_e]>}

Figure2. Example of CTNN

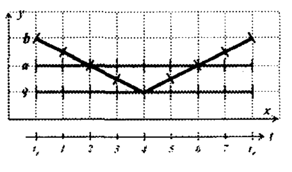


Figure3. Using Detection of Intersection and Symmetrical Movement

If we use simple approach to get this result, we must calculate distance between query and data objects at all evaluation time, from t_s to t_e, gradually (total 8 times).This may be incurred vast overhead. To reduce the number of time, we detect the intersections and do the symmetrical movement of line. Consider figure 3.

We form the half-plane based on query segments, and gather all data objects in one side space using symmetrical movement. And then we retrieve the intersection between segments (both query and data), and calculate distance between query and data objects at each evaluation time (start time of each segment, and intersection times). From these processes, we can find the result by less calculation than before (total 4 times). For more reducing, we must choose segments which will move. If we move all segments of one side to opposite side, it need much time. Therefore, we must decide what segments are moved, as well as when and where move them. We find intersection points using the sweep line algorithm. We regard object segments as lines in spatial environment. And then, we check whether objects are changed their position order at each endpoint interval of segments. For example, If *a* and *b* have the position order *a*→*b* at first initial evaluation time and have *b*→*a* at second one, then intersection occurred between *a* and *b*. We store the coordinate of intersection, intersection time and *id*s in intersection list *IL*. Here, we don't know yet whether they really intersect at any time. We assumed that objects which exist within a 1*m* diameter are occurred intersect. In this case, the time when object is putted on center of this diameter is the intersection time. Sometimes finding intersection objects may not intersect really. That is, for example, *a* reach on intersection point i_I at time 5 and *b* reach at time 8. But NN object may be changed at time 5 or 8. So, we regard the intersection time as earlier time putting on that position. As above example, i_I's intersection time is time 5. Following table 4 shows the rest process of CTNN. In the first stage, *TL* contains only initial evaluation times and *IL* contains candidate extra evaluation times. Here, *P* is all living data objects at any time.

Table4. Processing of CTNN

```
: For returned segments from temporal filtering step.
: return the result value from TL.

1. At first initial evaluation time $t_s$, separate $P$ into two groups
   A. If □$q$ move ($+\Delta x$ and $+\Delta y$), or ($+\Delta x$ and $-\Delta y$) $\Rightarrow$ objects which
      have $y_P \le y_q$ or $x_P \ge x_q$ are stored in list U. Otherwise, in list H.
   B. If □$q$ move ($-\Delta x$ and $+\Delta y$), or ($-\Delta x$ and $-\Delta y$) $\Rightarrow$ objects which
      have $y_P \le y_q$ or $x_P \le x_q$ are stored in list U. Otherwise, in H.
2. If first intersection $i_I$ at time $t_{iI}$ in IL belongs to H, calculate $|q-P_H|^{ts}$
   at $t_s$ (assume this case in here. H becomes the standard space,
   and U becomes the space to move). Otherwise, calculate $|q-P_U|^{ts}$.
3. Choose one object which $|q-P_H|^{ts} = mindist$, as NN object $p$.
   A. If there are several objects which have mindist, through the
      calculation of displacement, choose final one.
   B. If no result is returned from calculation, an object which has
      the smallest slope becomes $p$.
4. Among $P_U$, objects which have $|y_{iI}-y_q|^{tiI} \ge |y_{PU}-y_q|^{tiI}$ or $|x_{iI}-x_q|^{tiI} \le |x_{PU}-x_q|^{tiI}$
   move to opposite side on time interval [$t_s$, $t_{iI}$].
   Corresponding information in U also move to H, and become $P_U'$.
5. If there is $p'$ ($\in P_U'$) which $|p'-q|^{ts} < |p-q|^{ts}$, replace $p$ with $p'$.
   And then, inspect whether another intersection exist between □$p'$
   and □$P_H$ (except □$P_U'$). If intersection occurs, store them into IL.
6. Now, NN object in [$t_s$, $t_{iI}$] is $p$ which is selected in step 5. Within
   this interval,
   A. If there is another intersection $i_I'$ time $t_{iI}'$ in IL,
      I. Move $i_I'$ to H (if exist in U), also objects that consist of $i_I'$.
      II. Keep on (if exist in H) with interval [$t_s$, $t_{iI}$].
      ( Then, )
      I. $i_I'$ is made from current $p$ $\Rightarrow$ NN object in [$t_s$, $t_{iI}'$] is current $p$,
         and NN in [$t_{iI}'$, $t_{iI}$] is one which meet $p$ at $t_{iI}'$.
      II. $i_I'$ is not made from current $p$, ignore $i_I'$.
   B. If there is one of endpoints $t_I$ in TL and this point created by
      insertion of new object $p_I$
      I. Inspect whether $t_I$ exist in U or H.
         1. If $t_I$ be in U, move this to H with interval [$t_s$, $t_{iI}$], also object
            that intersect with $t_I$ in U as well as that intersect point.
         2. If $t_I$ be in H, keep on.
      II. Compare $|p_I-q|^{tI}$ and $|p-q|^{tI}$
```

1. If p_I is nearer than $p \Rightarrow$ NN object in $[t_s, t_I]$ is current p, and NN object in $[t_I, t_{il}]$ is p_I.
2. Otherwise, p_I is ignored, and inspection of intersection between □p_I and other things is not demanded.

C. If there is one of endpoints t_I and this created by deletion,

I. If any object except p is deleted, this point is ignored.

II. If p is deleted, an object which $|q-P_H|^{tI} = mindist$ becomes NN object.

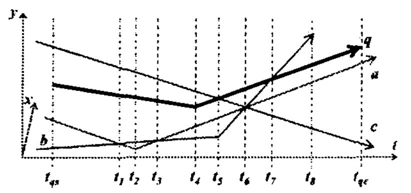
D. Until arrive at t_{il}, repeat step 6.

7. From t_{il} to i_k(the first intersection after t_{il}, exist in *IL*), repeat step 6. End time of each applying interval (will be applied in step 6) is initial intersection in *IL*. If the intersection point that is be end point of any applying interval exist in U, move this intersection and objects that consist of this intersection to H.
8. Until arrive at t_e, repeat step 6~7, and whenever information of NN object is settled, store NN object's id and valid time interval into TL, and update. Then at t_e, return whole information in TL as result value.

We use the coordinate list *CL* to compare coordinate between objects easily and quickly without extra calculation or scan. We also use the intersection list *IL* to select necessary intersection quickly, and use time list *TL* to store finding NN object information and to return them at last time point t_e. In CTNN information for endpoints of segments is not changed, because trajectory information is known already. Although it may seem that employment of three lists waste the space and require extra cost, they require few update and occupy somewhat little space than your thinking. What is more, if we find NN object p within $[t_i, t_{i+1}]$, objects which meet with p at t_{i+1} become candidate NN objects in next subsequence interval. At first evaluation time point t_s, we calculate distance between query object and all living data in one side at that time. And then find NN object p. Next, to find objects which meet with p at second evaluation time is only needed. And we can find information of intersection object easily in *IL*. Therefore, we can reduce unnecessary and duplicate calculation as well as number of data that will calculate.

D. Example processing of CTNN

Figure 4 is the result through the sweep line algorithm.



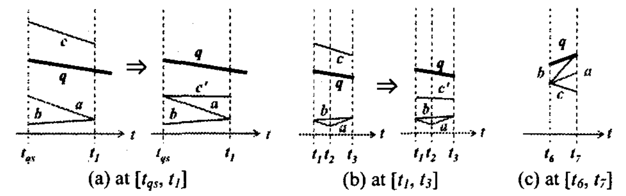
$CL = \{<a_1, (x_{a1}, y_{a1}), t_{qs}>, <b_1, (x_{b1}, y_{b1}), t_{qs}>, <c_1, (x_{c1}, y_{c1}), t_{qs}>, <a_2, (x_{a2}, y_{a2}), t_2>, <b_2, (x_{b2}, y_{b2}), t_5>, <b_3, (x_{b3}, y_{b3}), t_6> \dots\}$

$TL = \{ [t_{qs}, t_1], [t_1, t_2], [t_2, t_3], \dots, [t_8, t_{qe}] \}$

$IL = \{<t_1, (a, b)>, <t_3, (a, b)>, <t_5, (c, q)>, <t_6, (a, b, c)>, <t_7, (b, q)>\}$

Figure4. Result of detecting intersection

At t_s, Object a and b belong to lower side, list U, and c be in list H. The first intersection t_I is contained in U, so at t_{qs}, calculate distance between a, b and q, and then a become *candidate* in time interval $[t_{qs}, t_I]$. Next, c which is contained in H, have less coordinate value than t_I, so move it to U with interval $[t_{qs}, t_I]$. In U, $|c-q|^{tqs} = |a-q|^{tqs}$ and c have nearer segment than a on query. So, NN object in $[t_{qs}, t_I]$ become c (figure 5.a).



(a) at $[t_{qs}, t_I]$ (b) at $[t_I, t_3]$ (c) at $[t_6, t_7]$

Figure 5. Calculation of NN Object in Each Subinterval

Now, grasp whether c intersect with a or b. There is no intersection with c until t_I, so c remains candidate NN object until next intersection t_3. And also c is kept as NN object until t_6 (figure 5.b). At t_6, intersection is occurred by a, b, and c. From there, b is the NN object (figure 5.c). Like this, we calculate NN object until t_{qs}, and return the result from *TL* finally.

V. Conclusions

In this paper, we proposed a novel NN query technique that is suitable when both query and data objects are MOs. NN query in location-based applications must be able to find objects which move nearby trajectory of query object. Our proposed technique, CTNN consider the direction, velocity, and slope information of query object and data objects, and then using this information find exact and valid NN object continuously over the whole query time. CTNN technique can be used on navigational or traffic control systems as well as extended to k-NN query easily. CTNN may have some computational overhead when query is evaluated at the first time. But, at other time points, this needs the minimum comparison, update, and calculation. So, we can prevent unnecessary and repeated calculation and save the cost. We will propose other CTNN techniques to improve the performance in next version of paper, Approximate and Dynamic, and also show the evaluation of CTNN technique, and verify the superiority of ours.

References

[1] N. Roussopoulos, S. Kelley, F. Vincent, Nearest Neighbor Queries, SIGMOD Conference, pp.71~79, 1995

[2] K. Porkaew, I. Lazaridis, S. Mehrotra, Querying Mobile Objects in Spatio-Temporal Databases, SSTD, pp.59~78, 2001

[3] G. Kollios, D. Gunopulos, V. J. Tsotras, Nearest Neighbor Queries in a Mobile Environment, Spatio-Temporal Database Management, pp.119~134, 1999

[4] R. Benetis, C. S. Jensen, G. Karciauskas, S. Salenis, Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects, IDEAS, pp.44~53, 2002

[5] Y. Tao, D. Papadias, Spatial Queries in Dynamic Environments, TODS, pp.101~139, 2003