Nearest Neighbor Query Processing Techniques in Location-Aware Environment

SangHo Kim, BoYoon Choi, Keun Ho Ryu
Database Laboratory, Computer Science of Chungbuk National University
Cheongju, Chungbuk, 361-763, Korea
{shkim, bychoi, khryu}@dblab.cbu.ac.kr

Kwang Woo Nam, Jong Hyun Park
Spatial Information Electronic Center of Electronics and Telecommunications Research Institute (ETRI)
No. 161, Gajeong Dong, Usung Gu, Daejun, Chungnam, Korea
{kwnam, jhp}@etri.re.kr

Abstract: Some previous works for nearest neighbor (NN) query processing technique can treat a case that query/data are both moving objects. However, they cannot find exact result owing to vagueness of criterion. In order to escape their limitations and get exact result, we propose new NN query techniques, exact CTNN (continuous trajectory NN) query, approximate CTNN query, and dynamic CTNN query. These are all superior to pervious works, by reducing of number of calculation, considering of trajectory information, and using of continuous query concept. Using these techniques, we can solve any situations and types of NN query in location-aware environment.

Keywords: nearest neighbor query, moving, location-aware.

1. Introduction

Many GIS applications, such as air/vehicle traffic control system and meteorological observation system, treat moving objects and use variable query processing techniques. Among them, NN query is one of popular techniques, which finds object(s) that have nearest position from query. Many previous works for NN technique are applied to the case that query/data objects are moving/static or static/static objects. Some of them can process the case that query/data objects are both moving objects. But, they have still some limitations, for example, vagueness of criterion to choose NN object when there are many candidates. To conquer these limitations and get exact result, we propose new techniques - exact, approximate, and dynamic CTNN query. Exact CTNN query is nature foundation approach, and makes it possible that get the exact result. Although approximate CTNN query may have less exact result than exact CTNN, it has less calculation time cost. There are trade-off between cost and exactness of this approach. Dynamic CTNN query get exact result dynamically without any limitation.

Our techniques use trajectory information of objects to get exact result, and concept of *continuous query* [1] to get the timely response continuously over query. If we want to find one NN object while there are some candidates that have nearest distance simultaneously at any position of query trajectory, then previous works return result with randomly detecting. So, they may yield inaccurate or immediately changing (sometimes, these are meaningless in real world) result. If we consider trajectory information of objects, we can avoid this situation and get the exact result.

The rest of this paper is organized as follows. Related works are discussed in Section 2. Section 3 shows assumptions and notations. Our proposed techniques are described in Section 4. We briefly show evaluation of our techniques in Section 5, and then conclude this paper and present future works in Section 6.

2. Related Works

The cases of NN query processing that query/data objects are both moving objects are shown in [2, 3, 4]. Kollios[2] uses *duality transform* technique, but this is useless on more than two dimensions. Benetis[3] calculates distance between moving objects by using differential equation at periodical instant time on three dimensions, but it suffers from usual drawbacks of sampling. Tao[4] proposes continuous NN query technique. Though it finds continuously time points that result get changed on query, only k-NN query processing is possible.

Although [4] can solve NN query almost exactly, still have problem to get exact result. So we add considering of trajectory information to concept of continuous query. Use of continuous concept is little different between ours and [4], but its basic concept is same. Besides we can solve limitations of previous works.

3. Assumptions and Notations

We assume that moving objects are point objects and change only their position continuously over time. Information of moving objects is detected periodically and is stored in database with triple $<id, (x_i, y_i), t>$. This means that object *id* is located on (x_i, y_i) coordinate at time t. In terms of geography or geometry, the movement path is called trajectory, and this is represented by polylines, i.e., set of separate and sequential segments. Each segment connects two consecutive data points. These points are chosen when update (insertion, deletion,

This work was supported in part of University IT Research Center Project , KOSEF RRC Project (Cheongju Univ. ICRC) , and Spatial Information Electronic Center of ETRI in Korea

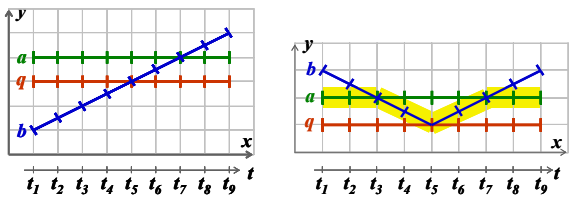
change of velocity or direction) is occurred. So each object has a constant velocity on one segment. In exact CTNN and approximate CTNN, we assume the whole trajectory information of data and query is predefined. So, it can be applied to a case as bus, train, and airplane. In this case, we can predict their trajectory, and also in case of query, user can input his/her expected trajectory. Following table 1 shows some notations.

Table 1. Notations

$\square a_i$	ith segment of object a ($i \geq 1$, integer)
x_{ab} (y_{ab})	absolute of difference value between a, b on x (y) axis
Δx (Δy)	moving distance on x (y) axis per unit time interval
$\lvert a\text{-}b \rvert^t$	$=\sqrt{(x_{ab}{}^t)^2+(y_{ab}{}^t)^2}$: distance between $\square a$ and $\square b$ at time t
$/p$	$=\Delta y \div \Delta x$: slope of segment p
∂p	$=\sqrt{\Delta x^2 + \Delta y^2}$: displacement of p
In case of $/p$ and ∂p , If both Δx and Δy have (+) value, result also have (+) value. Otherwise, result have (-) value.	

4. Continuous Trajectory NN Query

Basically, CTNN query contain three steps of calculation. First, we detect endpoints time of segments and store interval that is made of two consecutive detecting points, in time list TL. For simplicity, we call this *evaluation time*. Second, we find intersection points between all segments and store them in intersection list IL. In the real world, if objects have same coordinate at same time, this means collision and these cases hardly occur. So, if distance between objects is less than 1 meter, we regard these as intersect or meet. In order to get more exact result, we use symmetrical movement of data segments over query before calculating intersections. We show reason of this later. These steps are applied to both data and query trajectories. And all time points (except last evaluation time) become expectation points that changing of NN may be occurred. From these points, we can maintain concept of continuous query. Third, we calculate distance between query and data points at above points, and store object that has nearest distance in TL. When we want to find 1NN object, if there are many candidates that have nearest distance simultaneously, we choose one using *calculation of displacement* (table 2).



(a) Original information (b) After symmetrical movement

Fig. 1. Simple example of CTNN

For example, in fig. 1, assume that there are data objects <a, (1, 4), (9, 4), [t_1, t_9]>, <b, (1, 1), (9, 5), [t_1, t_9]> and query <q, (1, 3), (9, 3), [t_1, t_9]>. Evaluation times are t_1 and t_9, so initial TL={[t_1, t_9]}. Gather data segments in one space over query trajectory (figure 1(b)) and find intersection points, so IL={t_3, t_5, t_7}. Now, we calculate distance between query and data points at t_1, t_3, t_5, t_7. At

Table 2. 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

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

(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)

t_1, NN object is a and TL={<a, [t_1, t_9]>}. At t_3, NN object is changed to b and TL is updated as {<a, [t_1, t_3]>, <b, [t_3, t_9]>}. At t_5, still b is a NN object, so ignore. At t_7, again NN object = a, and TL is updated as {<a, [t_1, t_3]>, <b, [t_3, t_7]>, <a, [t_7, t_9]>}. Finally, CTNN returns TL.

1) Exact CTNN query

Exact CTNN query follows above basic calculations with some additional and changing of processing steps.

1 : Detect evaluation times and intersection times.

2 : At first evaluation time e_1, separate segments P in two groups over $q \rightarrow$ L(below) / H(above).

3 : Assume first intersection time i_1 belongs in H.
At e_1, calculate $\lvert q - P \rvert$ that belong in H.
Then, find 1NN object as p. Now, TL={<p, [e_1, i_1]>}

4 : At i_1, choose objects which belong in U and have nearer distance than $\lvert i_1 - q \rvert$. Then move them into H.

5 : At e_1, if there are any point which has nearer distance from q than p of Step3, then change p.

6 : If changed NN object make new intersection i_1' in [e_1, i_1], TL={<p, [e_1, i_1']>, <p_1, [i_1', i_1]>}. (p_1 meet with p at i_1').

7 : If there are another evaluation time points in [e_1, i_1], only pass Step5 and Step6.

8 : At i_1, start with p, repeat only Step4 ~ Step7, Until next intersection point of Step1.

2) Approximate CTNN query

We also propose approximate CTNN query technique to reduce calculation time cost of exact CTNN query, with submitting of a little error of result. If you allow more time cost, we may get exact result (of course, this need less time than exact CTNN query).

Approximate CTNN query use circle to find candidates. We call this *hunt circle*. This technique does not need symmetrical movement of segments.

Following is steps of approximate CTNN query.

1 : Detect evaluation times and intersection times.

2 : At first evaluation time e_1, calculate $|q - P|$.
Then find 1NN object as p.

3 : At e_1, Draw hunt circle h
(diameter is $|q - p|$, center point is middle of line qp)

4 : Find segments which intersect with h

4.1 : If there are no segments, TL={<p, [e_1, e_2]>}.

4.2 : If p_1 intersect with h at e_1', and p_1 have nearer distance than p, Then TL={<p, [e_1, e_1']>, <p_1, [e_1', e_2]>}.

4.3 : If i_1 is contained in h, and have nearer distance than p, Choose one of objects (that make i_1) as NN.
TL={<p, [e_1, i_1]>, <NN, [i_1, e_2]>}.

5 : Extend h into last point of first query segment.
Repeat Step4

6 : On each segment of query trajectory,
Repeat Step3 ~ Step5

3) Dynamic CTNN query

The above techniques have assumption that trajectory information is known already. But, Dynamic CTNN query have not this assumption.

Most previous works for NN query processing use index structure, such as TPR-tree, 3DR-tree, R-tree, and etc., for efficient and less consumptive finding and calculation. Our approaches also have better performance when use index. Especially, dynamic CTNN query have most superior effect by using index, because this have frequent updates by dynamic characteristics of data insertion or deletion.

Here, we only show basic processing steps of dynamic CTNN query with assumption that use index structure.

Given query interval (point) from (at) e_1,

1 : At e_1, Calculate distance between query and objects.

2 : Choose one object that has nearest distance, as p.

3 : If there are many candidates,
Wait until next insertion of their information,
Then choose one by calculation of displacement.
TL={<p, [e_1, e_2]>}.
(e_2 : is created by query (is created by insertion of data))

4 : At second insertion time of data, we can make first segment of data.
Then, find intersection using *calculation of displacement*.

5 : If there are any intersection i_1 that is made of p and p',
TL={<p, [e_1, i_1]>, <p', [i_1, e_2]>}.
Other intersections are ignored.

6 : Before time of i_1, if any data point p_1 is newly inserted at e_1', and have nearer distance than p,
Then, TL={<p, [e_1, e_1']>, <p_1, [e_1', e_2]>}.

7 : Before time of i_1, if any data point p_1 is deleted at e_1', and this is NN object at e_1',
Then, calculate new NN object (first candidate is NN of previous interval).

8 : From i_1, with p, Repeat Step3 ~ Step7

In this query, we can return TL when NN object is chosen surely or query is ended, selectively.

5. Analysis of Performance

In fig. 1, assume we calculate NN at evaluation time and intersection points without step of symmetrical movement (as fig. 1(a)). To get exact result, we calculate additionally at t_2, t_3, and t_4, and may have 3 more meaninglessness update (Mean that after calculating, there is no changing of NN) of TL. If we use symmetrical movement, it is enough that we have 1 meaninglessness update at t_5. In case of exact CTNN query, we separate all data into ABOVE or BELOW. Then, move the segments selectively and partially, and calculate distance between one side's all data and query. In case of previous works, they calculate distance between all data and query, so we reduce the number of calculation than previous one. Also we separate intersection point and evaluation point in two lists to reduce number of list update. It may looks like unnecessary division, but we found that it make better performance than others.

We may have much calculation time to find the first NN object. But, at all other time points, we don't need calculation of distance any longer, except occurring of deletion or newly insertion. We only check what object is met with p that is NN object until now. If many objects are met with p, only do calculation of displacement, and find exact 1NN object. Based on analysis, we can be convinced that our approaches are far superior to others.

6. Conclusion

We proposed new NN query techniques that are suitable when query/data are both moving objects. NN query in location-aware environment must be able to find objects which move nearby over query. Our approaches can satisfy this condition as we find exact and valid NN objects continuously over the whole query time with considering of direction, velocity, and slope of objects. Also, ours are appropriately applied to all cases. That is, these have no concern with type of data/query (static or moving) and trajectory (unknown or known in advance). Moreover, these can be extended to k-NN query easily.

We will show cases that process CTNNs with index structure in later version. And we are improving our approaches, so we can soon show better performance. Also, we will develop extending CTNN versions that can be applied to the case that moving objects have region.

References

[1] A. P. Sistla, O. Wolfson, S. Chamberlain, S. Dao, 1997. Modeling and Querying Moving Objects, *ICDE*, pp.422~432.

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

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

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