http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (5): 900-907, 2006
ISSN 1812-5638

Continuous Monitoring of Predictive Spatio-temporal Range Query

[1]Wei Zhang, [1]Jianzhong Li, Shengfei Shi and [2]Haiwei Pan
[1]School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China
[2]Department of Computer Science, Harbin Engineering University, Harbin, China

Abstract: The predictive Spatio-temporal range query returns a set of moving objects that will appear in a spatial query range during a future temporal query interval. A novel approach is proposed to continuously monitor the changes in the results of predictive queries as objects moves. Two index structures are designed to increase the efficiency of query monitoring. The experimental evaluation shows that the proposed method can efficiently monitor massive predictive Spatio-temporal range queries over moving objects.

Key words: Spatial databases, query processing, index methods, location-dependent

INTRODUCTION

As the development of positioning technology and wireless communication, moving objects database has received considerable attention in recent years. The predictive Spatio-temporal range query is one of the most common queries in moving objects database. It returns a set of moving objects that will appear in a spatial query range during a future temporal query interval. Users often make decisions based on the results of predictive queries. For example, by querying the vehicles which will possibly appear in some hot regions after 50 min, the operator of an intelligent transportation system can take action to avoid future traffic jam in those areas; by querying the cars or people that will possibly move nearby a supermarket after 30 ~ 40 min, the manager of the supermarket can send electronic coupons through location-dependent information service to attract those potential customers.

The results of predictive Spatio-temporal range queries are computed according to the predicted future location of moving objects. There are three methods for predicting the future location of moving objects. (1) According to the latest reported location and velocity of objects, the future location is computed by linear motion function (Saltenis *et al*., 2000; Tao *et al*., 2003, Hadjieleftheriou *et al*.,2003). (2) According to the location of objects in recent k instants, the future location is computed by linear recursion function (Tao *et al*., 2004). (3) According to the max speed of a moving object, the future location range of the object can be estimated. If the query range overlaps the future location range of the object, the probability that the object will move to the overlapping region can be computed according to the distribution function of object's future location. Reynold *et al*., propose a probabilistic method to process predictive k-NN queries (Cheng *et al*., 2003, 2004). Their query processing method assumes object's future location is uniformly distributed in its future location range. Since the prediction of objects' future location is based on their latest reported location and velocity, the integrality and correctness of query results must be influenced by the update of objects' location or velocity. Therefore, the results of predictive queries must be continuously updated in accordance with the variable information of moving objects, especially for the queries about long-term future state. This requires that database system can efficiently detect and update the changes of returned results which are influenced by the movement of objects. However, most of current research works only focus on processing of predictive Spatio-temporal range queries and do not consider the continuous monitoring of the massive predictive queries.

The native approach for continuous monitoring of predictive queries is to repeatedly reprocess all queries after a regular interval. However, this scheme has at least two drawbacks. Firstly, it can not detect the changes of returned results in real time. Secondly, since the cost of continuous monitoring is in proportion to the number of queries, this naïve approach is not suitable for processing massive queries.

In the study, we proposed a novel approach for continuous monitoring of predictive Spatio-temporal range queries. Our method separates the processing of new queries from continuous monitoring of processed queries. Two index structures are designed. TPMR-Tree stores the constantly changed location of moving objects and supports the processing of new queries. CQ-Index

Corresponding Author: Wei Zhang, School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China

records all processed queries and support continuous monitoring of query results. It should be noted that our method is a general framework which can support different kinds of location predicting methods.

There are many research works from different aspects on processing of predictive spatial-temporal queries. Saltenis *et al.* (2000) and Tao *et al.* (2003) proposed an R*-tree based index structure. TPR*-Tree is one of the best index structure to support the processing of predictive spatial-temporal range queries (Tao *et al.*, 2003). We compare our approach with TPR*-Tree index in experimental evaluation. Hadjieleftheriou *et al.* (2003), propose a predictive range aggregation query processing method. All above query processing approaches employ linear motion function to predict the future location of moving objects. Tao *et al.* (2004) use the linear recursion function to predict the future location of moving objects based on their recent k positions. However, this method requires that each moving object has computation and storage resource, which are not satisfied applicable in some applications. In addition, since it is difficult to measure the precise location of moving objects in some applications, Cheng *et al.* (2003 and 2004) propose several kinds of probabilistic query over imprecise data and study the processing of predictive k-NN queries based on probabilistic method. Nevertheless, all queries in Cheng *et al.* (2003) are defined to evaluate the probabilistic queries about a time instant. In fact, our continuous query monitoring method can support the processing of probabilistic Spatio-temporal range queries too.

Recently, continuous query processing over current location of moving objects has also been studied. By Mouratidis *et al.* (2005), Xiong *et al.* (2005), Yu *et al.* (2005), three continuous k-NN query monitoring approaches are proposed. All these methods are based on main memory grid index. However, these continuous query monitoring methods can not process predictive query. In the paper, we study continuous monitoring of predictive Spatio-temporal range query. Our approach supports not only the linear motion function based predicting method, but also the probabilistic model based predicting method.

PROBLEM DEFINITION AND SOLUTION OVERVIEW

We assume objects move in a 2D space S = $[L_1, L_2]\times[U_1, U_2]$. A moving object reports its location to the server in a fixed rate or sends the update messages when its velocity is changed. Let t_u represent the latest time when a moving object O reports its location and $O.P(t) = (O.P_x(t), O.P_y(t))$ represent the location of O at t. $O.V_{max}$ is the maximum speed of O. The location of an object between two successive location update is computed by the linear interpolation. Table 1 gives the description of symbols.

Table 1: Description of symbols

Symbol	Description
t_c	Current time
t_u	The latest time when a moving object reports its location
t_{sq}	The time when a PRQ is submitted
t_q, T	The query time or query interval of a PRQ
O.P(t)	The location of objects O at t
$O.V_{max}$	The maximum speed of O

Definition 1: Given a future time $t > t_u$, the possible location space of a moving object O at t is a round area which is centered at $O.P(t_u)$ with radius $O.V_{max}\times(t-t_u)$, denoted as O.PLS(t).

If the object's velocity $O.V = (O.V_x, O.V_y)$ is available, the current location of O can be computed by the linear motion function. Then, Given any future time $t > t_c$, O.PLS(t) can be calculated by the equation

$$(x-(O.P_x(t_u) + O.V_x (t_c-t_u))) (y-(O.P_y(t_u) + O.V_y (t_c-t_u)))-O.V_{max} (t_f-t_c) = 0.$$

It should be noted that the velocity of a moving object is not a prerequisite of the continuous query monitoring. According to the max speed of a moving object, definition 1 can be directly applied to compute the possible location space.

Definition 2: Mobile rectangle $MR(R, V_R, t, t_s)$ is a rectangle area whose size and center can change over time, where $R = [x_1, x_2]\times[y_1, y_2]$ is the location of mobile rectangle at t_s, $t \geq t_s$ is the time variable, $V_R = < v_l, v_r, v_u, v_d >$ is the velocity vector, v_l and v_r is the horizontal speed of left and right borderlines of R, v_u and v_d is the vertical speed of up and down borderlines of R, $MR(R, V_R, t, t_s) = [x_1 + v_l (t-t_s), x_2 + v_r(t-t_s)]\times[y_1 + v_d(t-t_s), y_2 + v_u(t-t_s)]$.

Figure 1 gives an example of mobile rectangle, the initial state at $t_s = 0$ is $R = [1, 2]\times[5, 4]$ and $v_l = v_r = v_u = v_d = v$, then the mobile rectangle at t = 5 is $MR(R, V_R, 5, 0) = [(1 + 5v), (2+5v)]\times[(5 + 5v), (4 + 5v)]$. If v = 0, the

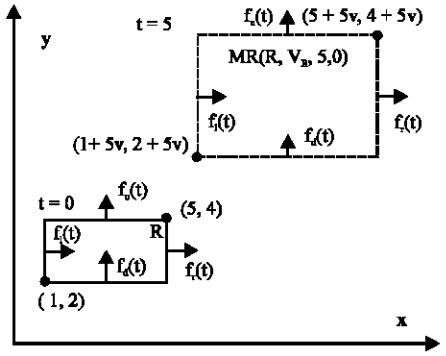

Fig. 1: An example of mobile rectangle

rectangle becomes a static region, i.e., $MR(R, V_R, t, t_s) = R$ for any t. The mobile rectangle can represent a mobile query range during a query time interval.

Definition 3: Given a future interval $T = [t_l, t_h]$ and mobile rectangle $MR(R, V_R, t, t_l)$, predictive Spatio-temporal range query returns the set of objects that will move inside $MR(R, V_R, t, t_l)$ during T, i.e., $PRQ(T, MR(R, V_R, t, t_l)) = \{O_i | \exists t \in T, O_i.P(t) \in MR(R, V_R, t, t_l)\}$.

Definition 4: The candidates set of the query $PRQ(T, MR(R, V_R, t, t_l))$ is a set of objects containing all objects that are possibly satisfied the query. i.e., $CP = \{O_i \mid \exists t \in T, O_i.PLS(t) \cap MR(R, V_R, t, t_l) \neq \varnothing\}$.

Given a query $PRQ(T, MR(R, V_R, t, t_l))$ submitted at t_{iq}, for any t_e ($t_{iq} \leq t_e \leq t_h$), the study focuses on continuous monitoring of $PRQ(T, MR(R, V_R, t, t_l))$ during $[t_{iq}, t_e]$. t_e is called the effective time of a query. If current time $t_c = t_e$, the query is a valid query and needs to be continuously monitored, otherwise, the query is a invalid query which can be discarded.

Figure 2 gives an overview of the continuous query monitoring system. The study employs two schemes to process predictive queries. (1) If $PRQ(T, MR(R, V_R, t, t_l))$ is a new submitted query, at first, the candidates set of the query are retrieved. Then, the future location of objects in the candidates set is computed according to a location predicting method. Finally, objects whose predicted future location will satisfy the query are returned. In order to increase the efficiency of retrieving candidates set, we propose an index structure for moving objects, TPMR-Tree. (2) For all processed queries, the candidates set of the queries are continuous monitored. After the location of objects is updated, if the result of any processed query is changed, the updated result will returned to corresponding user. In order to increase the efficiency of continuous monitoring, a query index, CQ-Index, is designed. According to the candidates set of a processed query, a tuple (object id, query id) is inserted into CQ-Index. After the location of a moving object is updated in TPMR-Tree, queries that are possibly influenced by the object are determined by querying CQ-Index. Then, the future location of the object is computed and is checked against the retrieved queries. If the object will not satisfy any of the retrieved queries, the result of the query in is updated in corresponding entry in the query table.

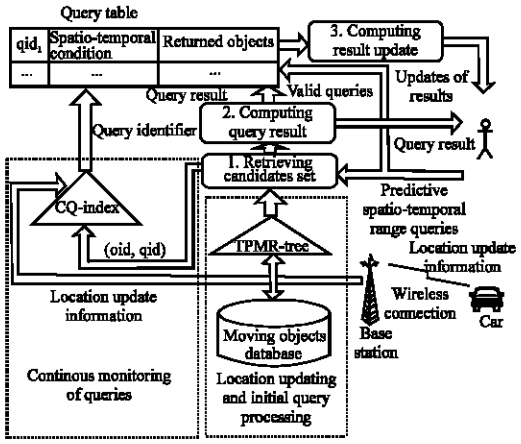

Fig. 2: Continuous query monitoring system

Our continuous query monitoring approach has two advantages. (1) Generality. Since the retrieving of candidates set does not depend on any specific location predicting method, the proposed approach can support the processing of predictive queries according to different location predicting methods. (2) Load balancing. TPMR-Tree is used to index the location of moving objects and support the retrieving of candidates set. CQ-Index is used to index processed queries by objects in their candidates set and support continuous monitoring of the processed queries. Then, location updating of moving objects can be processed in parallel with continuous monitoring of queries. By dividing the workload of location updating, candidates set retrieving and continuous query monitoring, this approach can balance the workload and improve the efficiency of system.

TPMR-TREE

TPMR-Tree (Time-Parameterized Max speed Rectangle Tree) is designed to efficiently index the location of moving objects and retrieve the candidates set of predictive Spatio-temporal range queries.

It is an extension of R*-Tree. Data in a leaf node is in the form of $(< O_1.P(t_1), O_1.V_{max}, t_1 >, < O_2.P(t_2), O_1.V_{max}, t_2 >, \ldots, < O_i.P(t_i), O_i.V_{max}, t_i >)$, where t_i is the latest time that object O_i reports its location. Data in a non-leaf is in the form of $(< MBR_1, T_1, V_1, ptr_1 >, < MBR_2, T_2, V_2, ptr_2 >, \ldots, < MBR_k, T_k, V_k, ptr_k >)$, where ptr_k is the pointer of a subtree or a leaf node, MBR_k is the minimum bounding rectangle of all moving objects that are stored in subtree or leaf node pointed by ptr_k, T_k is the minimum time interval that contains all location update times of all objects that are stored in subtree or leaf node pointed by ptr_k, V_k is the max speed of all objects that are stored in subtree or leaf node pointed by ptr_k. Figure 3 gives an example of TPMR-Tree. In the Fig. 3, V_1 is the max speed of all objects contains in MBR_1, i.e., $V_1 = Max\{O_1.V_{max}, O_2.V_{max}\}$, interval $[t_1, t_2]$ contains the location update time of all objects contains in MBR_1.

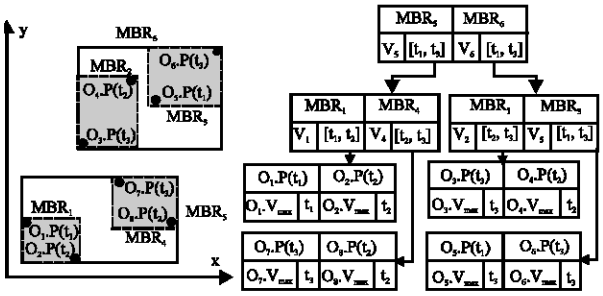

Fig. 3: An example of TPMR-tree

While processing the query PRQ(T, MR(R, V_R, t, t_l)) in TPMR-Tree, the query algorithm first computes the extent of each MBR or the possible location space of objects at every future time t during T, then, the candidate objects are retrieved if its possible location space will overlap the mobile rectangle MR(R, V_R, t, t_l) during T. Specifically, Starting from the root of the tree, level by level, the algorithm visits the subtree whose MBR will overlap MR(R, V_R, t, t_l)) during T. If current node is non-leaf node, the extent of the node's MBR during T can be computed according to the speed V in the node. The overlap state can be determined by the method used in (Saltenis *et al*., 2000). If a MBR overlaps the query range during T, the algorithm checks the subtree or the leaf node that is indexed by MBR. Since the possible location space of a moving object is a round region, while checking objects in a leaf node, the query algorithm uses two phases to determine a candidate of the query. For any O_i in a leaf node, firstly, the algorithm use the method that is same as the one used to check non-leaf node to determine whether the bounding rectangle of O_i.PLS(t) overlaps the query range during T. If they do not overlap, O_i is discarded; otherwise, the next checking phase is evoked to check the validation of O_i. The detail query processing algorithm is given below.

Algorithm PRQuery (Tr, T, MR(R, V_R, t, t_l))
Input Tr: The root node of TPMR-Tree
PRQ(T, MR(R, V_R, t, t_l)) : Predictive query ;
Output CP: Candidates set of the query
1. CP = ∅;
2. If (Tr is a leaf node)
3. CP = CP ∪ Chk_Leaf(Tr, T, MR(R, V_R, t, t_l));
4. Else { //Tr is a non leaf node
5. For all MBR_i in Tr
6. If (MBR_i overlaps MR(R, V_R, t, t_l) during T)
7. CP = CP ∪PRQuery(ptr_i, T, MR(R, V_R, t, t_l))
8. Return CP;

Algorithm Chk_Leaf (Node, T, MR(R, V_R, t, t_l))
Input Node : Leaf node of the tree
PRQ(T, MR(R, V_R, t, t_l)): Predictive query ;
Output C : the set of canddiates in Node
1. C = ∅;
2. For all O_i ∈ Node
//O_i possibly satisfy the query
3. T' = the interval that O_i.PLS(t) overlaps MR(R, V_R, t_l, t_l) during T;
4. If (Refine_Check (T', MR(R, V_R, t, t_l), O_i)
5. C = C ∪ O_i; //O_i satisfy the query
6. Return C;

The algorithm refine_check is used to determine whether a moving object in leaf node will satisfy a query. The main idea of the algorithm can be described as follow: (1) if T is a future time instant t_f, the algorithm only needs to check whether the query rectangle overlaps O.PLS(t_f). If the minimum distance between O.P(t_u) and MR(R, V_R, t, t_l) is greater than O.V_{max} (t_f-t_u), O can not satisfy the query. (2) If T is a time interval, by applying the same method for checking non-leaf node, the algorithm can compute the interval T'= [t_1, t_2] ⊆ T during which MR(R, V_R, t, t_l) overlaps the bounding rectangle of O.PLS(t). Then, the algorithm needs to verify if there exits a future time instant t_q∈T' at which O.PLS(t_q) overlaps MR(R, V_R, t_q, t_l). If t_q can be found, O is returned as a candidate. Figure 4 gives an example of computation in refine_check. Suppose O.PLS(t) does not overlaps MR(R, V_R, t, t_l) at t = t_1. As shown in Fig. 4, it can be proved that if there exists a instant t_q∈T' at which O.PLS(t_q) overlaps MR(R, V_R, t_q, t_l), the point which first reaches O.PLS(t_q) must be the point of MR(R, V_R, t, t)$_l$ that is nearest to O.PLS(t) at t . Suppose the coordinate of the point is (x(t), y(t)). The distance between the point and O.P(t_u) must be the radius of O.PLS(t_q). Then, we get a distance equation:

$$(x(t) - O.P_x(t_u))^2 + (y(t) - O.P_y(t_u))^2 = (O.V_{max}(t - t_u))^2$$

In Figure 4, the point that reaches O.PLS(t) is the left up corner of MR(R, V_R, t, t_l). Then, x(t) and y(t) can be presented in $x(t) = x(t_1) + v_l(t-t_1)$ and $y(t) = y(t_1) + v_u(t-t_1)$. By substituting them into above equation, we get a standard equation which is in the form of $At^2 + Bt + C = 0$, where

$A = (v_l^2 + v_u^2) - (O.V_{max})^2$;
$B = 2\bullet(v_l\bullet(x(t_1) - O.P_x(t_u)) + v_u\bullet(y(t_1) - O.P_y(t_u))) + 2\bullet t_u\bullet(O.V_{max})^2$;
$C = (x(t_1) - O.P_x(t_u))^2 + (y(t_1) - O.P_y(t_u))^2$
$-(t_u \bullet O.V_{max})^2$;

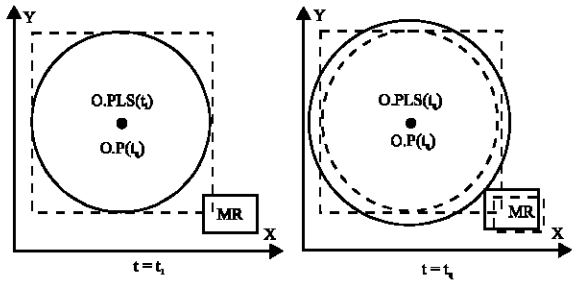

Fig. 4: An example of refine_check computation

If the equation has a root $t_q \in T' = [t_1, t_2]$, O satisfy the query, otherwise O is discarded. The detail algorithm is given below:

Algorithm Refine_Check (T', MR(R, V_R, t, t_l), O)
Input Mobile rectangle MR(R, V_R, t, t_l),
Overlap interval T' = [t_1, t_2], Object O;
Output Is O satisfy the query
1. If (O.PLS(t_1) ∩ MR(R, V_R, t_1, t_l) ≠ ∅
2. Return true; //O satisfy the query at t_1
3. If (t_1 = = t_2) // T' is a time instant
4. Return false;
5. Else // T' is a time interval
6. t_q = the positive root of distance equation;
7. If ($t_q \in T'$) Return true;
8. Else Return false;

The TPMR-Tree is constructed by invoking the insert algorithm to insert objects into the index. While inserting the location of objects into TPMR-Tree, the insert algorithm first computes the current extent of non-leaf node's MBR and then chooses a suitable subtree to insert the data. The MBR of a non-leaf node can be considered as a mobile rectangle. The expanding speed of MBR's borderlines is the max speed of objects in the MBR. Suppose location update interval in a non-leaf node is [t_m, t_n]. According to the expanding speed of MBR and (t_c-t_n), current extent of the non-leaf node can be computed. Then, the algorithm employs the strategy which is similar to the strategy proposed in (Tao *et al.*, 2003) to choose the subtree. If a subtree is chosen, the MBR, max speed and location update interval needs to be updated. If a leaf node, which is chosen to insert a new object, is full, some objects are picked out from the leaf node. The policy of choosing objects is to minimize the future extent of the MBR that covers the remaining object in the leaf node. The insert algorithm of TPMR-Tree adopts the "reinsert" scheme in R*-Tree to process those picked out objects. The algorithm first tries to insert those objects into the TPMR-Tree. If any node is overflowed, the node split method is invoked. The node splitting strategy is similar to that of TPR*-Tree.

The location update of TPMR-Tree is implemented by deleting the old location and inserting the new location of the object. The delete algorithm of TPMR-Tree is similar to that of R*-Tree. After the old location is deleted, the MBR, max speed and location update interval need to be updated too.

After retrieving the candidates set from TPMR-Tree, the candidate objects whose future location will satisfy the query are returned. While computing the future location of objects, any kinds of location predicting methods can be applied, e.g., predicting based on linear monition function. In order to efficiently monitor the query result, the processed queries are recorded in CQ-Index according to their candidates set. After an object updates its location in TPMR-Tree, the queries whose results are influenced by the object's location update can be found by searching CQ-Index with the object's identifier. According to definition 1, it can be proved that if $\exists t_q \in T$, O.PLS(t_q)∩MR(R, V_R, t_q, t_l) = O.PLS(t_q), O must satisfy the query PRQ(T, MR(R, V_R, t, t_l)). Therefore, those objects which will definitely satisfy a given query are not inserted into CQ-Index with the corresponding query.

CQ-INDEX

CQ-Index is designed to support continuous monitoring of massive predictive Spatio-temporal range queries (Fig. 5). It indexes all processed and valid queries (i.e., $t_c \le t_e$) by the identifiers of objects in the queries' candidates set. Error! Reference source not found. gives an example of CQ-Index. According to the figure, the objects, whose identifiers are 4 and 1, are in the candidates set of q_1 and q_3. By applying CQ-Index, we can monitor all query results that may be influence by location update of moving objects.

As shown in Error! Reference source not found., the structure of CQ-Index is similar to that of B-Tree. The data in a leaf node is in the form of (< oid_1, qtr_1 >, < oid_2, qtr_2 >, …, < oid_m, qtr_m >), where $oid_1 < oid_2 < \ldots < oid_m$, qtr_i is the

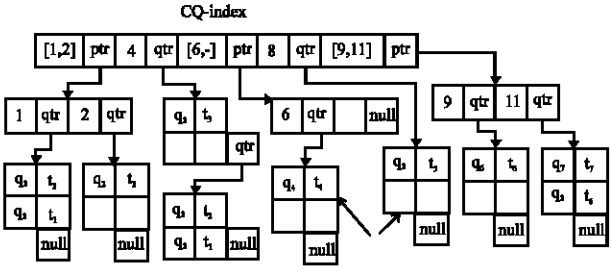

Fig. 5: An example of CQ-index

pointer of a query list. All queries whose results may be influenced by the object with identifier oid_i are stores in the query list pointed by qtr_i. The list pointed by qtr_i is in the form of ($< qid_1, t_1 >, < qid_2, t_2 >, \ldots, < qid_n, t_n >, qtr_{next}$), where qid_i and t_i are the identifier and effective time of a query, qtr_{next} is the pointer of next block of the list. If qtr_{next} is not null, the block pointed by it must be full of queries. The data in a non-leaf node is in the form of ($< (oid_{l\_1}, oid_{h\_1}), oid_1, qtr_1, ptr_1 >, < (oid_{l\_2}, oid_{h\_2}), oid_2, qtr_2, ptr_2 >, \ldots, < (oid_{l\_k}, oid_{h\_k}), oid_k, qtr_k, ptr_k >, < (oid_{l\_(k+1)}, oid_{h\_(k+1)}), ptr_{k+1} >$), where $oid_1 < oid_2 < \ldots < oid_k$ and $oid_i < oid_{l\_(i+1)} < oid_{h\_(i+1)} < oid_{(i+1)}$, qtr_k is the pointer of the query list recording all queries whose results may be influenced by the object with identifier oid_k, ptr_i points to the subtree with the key less than oid_i and ptr_{k-1} points to the subtree with the key in the range of $[oid_{l\_(k-1)}, oid_{h\_(k+1)}]$.

CQ-Index is constructed by inserting the processed queries into the index according to their candidates set. For all processed queries, an entry (oid, < qid, t_e >) can be generated according to the candidates set of the query identified by qid, where oid is the identifier of an object containing in the query's candidates set, t_e is the effective time of the query. The insert algorithm uses the method similar to that of B-Tree to insert the entry (oid, < qid, t_e >) into CQ-Index. If oid is found in a non-leaf node, the algorithm directly inserts qid into corresponding query list; otherwise, the query is inserted into a query list in leaf level.

When an object O updates its location, the query algorithm searches the CQ-Index by the identifier of O to find whether there is a query list indexed by O. If the query list is found, all queries whose results may be influence by the movement of O can be identified. By computing the new future location of O and checking against the queries, the query algorithm of CQ-Index can continuously report the updated results of predictive queries.

CQ-Index is maintained by a lazy-update method. While scanning queries in a query list, we use different operations to maintain the data in CQ-Index according to different conditions. (1) If an invalid query (i.e., $t_e > t_e$) is found, the query identifier is deleted from the list. (2) If an invalid query is found during inserting an entry in a block, the query identifier is deleted from the list. (3) If it can be determined that the possible location space of an object is fully contained (or outside) in spatial query range, the query is deleted from the list. Since the object must satisfy (or not satisfy) the given query. (4) If a query list, which is indexed by an object identifier in the leaf node, becomes empty after a query is deleted from the list, the object is deleted from the leaf node. (5) When the filled rate of a leaf node is less than 50% after an object is deleted, the node merging method, which is similar to that of B-Tree, is invoked. (6) When a query list which is connected to a non-leaf node becomes empty after the last query is deleted from the list, the object which is the key of the list does not remove from the index immediately. Objects in non-leaf node are deleted only when splitting a leaf node or merging nodes in the index.

RESULTS

All experiments run on a Linux system equipped with P IV 2.4 GHz processor and 512 MB memory. We use the trajectory data generator which is same as the one used in (Hadjieleftheriou *et al.*, 2003). Objects move in the road networks of Illinois (Fig. 6). The map of road networks is normalized to 1000×1000 km^2. The output of the generator is a set of trajectories.

Initially, 100,000 moving objects uniformly distributed in the space. At any time instant, the speed of objects is uniformly distributed in 60 ~ 120 km h^{-1} and the max speed of objects is set to 120 km h^{-1}. The location update rate is the number of objects whose location is updated during one minute as a fraction of all objects. In the experiment, the default location update rate is 50%, i.e., half of moving objects update their location during one minute. For all queries in form of PRQ(T, MR(R, V_R, t, t_l)), where R is uniformly distributed in the space and covers 0.25% of the space, $V_R = \{0, 0, 0, 0\}$. The query interval of all predictive queries is T = [30, 40] minutes and each query are continuous monitored for 10 min. In the figures given blow, we use CPRQ to denote our continuous query monitoring approach. Since there is no similar

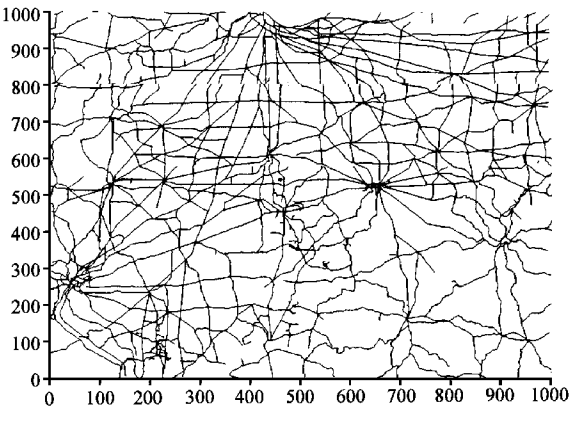

Fig. 6: Road networks of Illinois

research work, we compare CPRQ with the naive continuous query monitoring approach which uses TPR*-Tree.

The performance of TPMR-tree: Figure 7 compares the average I/O cost of processing 500 predictive queries by TPMR-Tree and TPR*-Tree. The disk block is set to 1 KB and buffer is set to 10 KB. The query interval is $[t_l, t_l + 10]$. The result shows that the I/O cost increases as t_l increases. Although the query processing cost of TPMR-Tree is a little higher than that of TPR*-Tree (28.3% on average), candidates sets of the queries can be retrieved by using TPMR-Tree while using TPR*-Tree only gets the objects which will satisfy the queries according to their linear motion function.

The effect of query scalability: Figure 8 compares the average I/O cost of continuous monitoring of the predictive queries by CPRQ and by TPR*-Tree. TPR*-Tree processes predictive queries according to the linear motion of objects. In order to support continuous monitoring of predictive queries, it must reprocess all

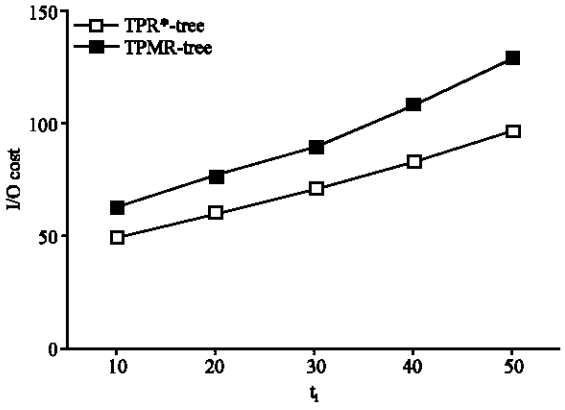

Fig. 7: I/O cost of initial query processing

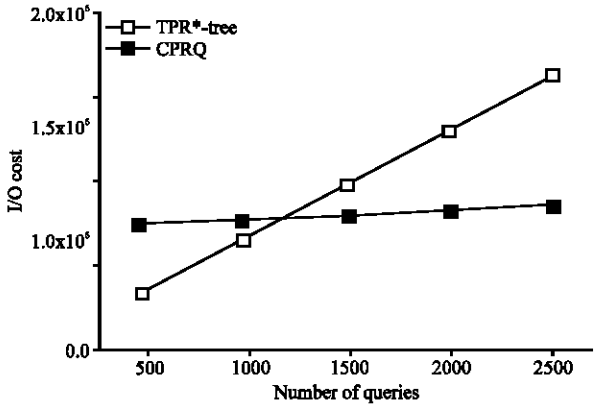

Fig. 8: The effect of query scalability

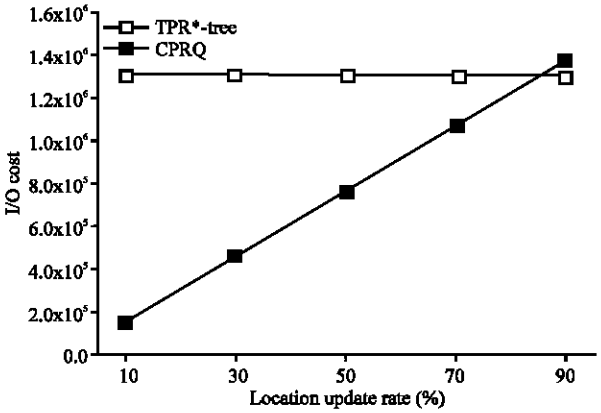

Fig. 9: The effect of location update rate

queries after a given time interval. In the experiment, the queries are reevaluated once per minute. While in our query continuous query monitoring approach, the result, which is influence by the location update of an object, can be computed immediately after the object moves. The figure shows that the cost is in proportion to the number of queries. The cost of continuous monitoring based on TPR*-Tree is more sensitive to the increase in number of queries. This reveals that CPRQ is more suitable for massive query monitoring.

The effect of location update rate: In order to reduce the communication cost, moving objects can update their location in different update rate, e.g., updating location only when the velocity changes (Saltenis *et al*., 2000, Tao *et al*., 2003; Hadjieleftheriou 2003) or reporting location when the distance between current location an latest reported location exceeds a given threshold (Tao *et al*., 2004). Then, the location update rate of moving objects is in the range of (0, 100%]. We vary the location update rate from 10 to 90%. Figure 9 evaluates the effect of location update rate on query monitoring cost. In the experiment, 2000 queries are continuous monitoring for 10 min by two different approaches. It shows that the naïve query monitoring method does not effected by location update rate, since this query monitoring scheme reprocesses all queries at each update time. Meanwhile, the cost of CPRQ increases linearly with the location update rate. Therefore, CPRQ is more suitable for monitoring the objects with lower location update rate.

CONCLUSIONS

Predictive Spatio-temporal range query is one of the most common queries in the moving objects database. We first address the problem on continuous monitoring of the predictive Spatio-temporal range queries and propose an

efficient query monitoring approach. The experimental evaluation shows that by employing two index structures designed in the study, the proposed method can efficiently monitor massive predictive queries.

REFERENCES

Cheng, R., D.V. Kalashnikov and S. Prabhakar, 2003. Evaluating probabilistic queries over imprecise data. In Proceedings of SIGMOD., pp: 551-562.

Cheng, R., D.V. Kalashnikov and S. Prabhakar, 2004. Querying imprecise data in moving object environments. IEEE Trans. Knowl. Data Eng., 16: 1112-1127.

Hadjieleftheriou, M., G. Kollios, D. Gunopulos and V.J. Tsotras, 2003. On-line discovery of dense areas in Spatio-temporal databases. In Proceedings of SSTD., pp: 306-324.

Mouratidis, K., D. Papadias and M. Hadjieleftheriou, 2005. Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor monitoring. In Proceedings of SIGMOD., pp: 634-645.

Saltenis, S., C.S. Jensen, S.T. Leutenegger and M.A. Lopez, 2000. Indexing the positions of continuously moving objects. In Proceedings of ACM SIGMOD., pp: 331-342.

Tao, Y., D. Papadias and S. Jimeng, 2003. The TPR*-Tree: An optimized Spatio-temporal access method for predictive queries. In Proceedings of VLDB., pp: 790-801.

Tao, Y., C. Faloutsos, D. Papadias and L. Bin, 2004. Prediction and indexing of moving objects with unknown fdmotion patterns. In Proceedings of SIGMOD., pp: 611-622.

Xiong, X., F. M. Mohamed and G.A. Walid, 2005. SEA-CNN: Scalable processing of continuous k-nearest neighbor queries in Spatio-temporal databases. In proceedings of ICDE., pp: 643-654.

Yu, X., K. Q. Pu and N. Koudas, 2005. Monitoring k-nearest neighbor queries over moving objects. In Proceedings of ICDE., pp: 631-642.