Time-Series Prediction with Applications to Traffic and Moving Objects Databases[*]

Bo Xu
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607, USA
boxu@cs.uic.edu

Ouri Wolfson
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607, USA
wolfson@cs.uic.edu

ABSTRACT

In this paper we explore the application of travel-speed prediction to query processing in Moving Objects Databases. We propose to revise the motion plans of moving objects using the predicted travel-speeds. This revision occurs before answering queries. We develop three methods of doing this. These methods differ in the time when the motion plans are revised, and which of them are revised. We analyze the three methods theoretically and experimentally.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications

General Terms

Algorithms, Performance

Keywords

Time-series Prediction, Spatio-temporal Query, Moving Objects Databases

1. INTRODUCTION

In this paper we investigate how travel-speed prediction can be incorporated in Moving Objects Databases (MOD). Such databases store, among others, the estimated (or expected) future motion plans of moving objects such as vehicles, pedestrians, aircraft, etc. We examine two types of queries enabled by a MOD. The first type are *point* queries that pertain to the motion of a single specified object. For example, "will the object O arrive at its destination by time t?". The second type are *range* queries that retrieve the moving objects that pass a given region during a given time interval. For example, "retrieve the objects (e.g. trucks) that are expected to be within two miles of a given warehouse W sometime between t_1 and t_2.".

Clearly, for vehicles moving on the road network, if t, t_1, t_2 are in the future, then the answers to the above queries depend on the traffic, or expected travel-speeds, on the roads. Observe further that what matters is not only the current travel-speeds, but also the travel-speeds that will exist from now until the future time t, t_1, or t_2. And this is exactly where time-series prediction comes in.

In this paper we assume that the future motion plan of a moving object is represented by a trajectory. This trajectory defines by a three-dimensional polyline the estimated (or expected) locations of the object at future time points. A natural way of incorporating travel-speed prediction is as follows. Whenever a speed update is received from sensors on a road segment B (indicating that, for example, the current average speed on B is 40 miles/hour), future speeds are predicted for B. The predicted speeds are used to update all the trajectories that travel B. We call this approach *Speed Update Triggered Revision (SUTR)*. SUTR creates a performance problem when speed updates are frequent or the size of the road network is large, since it requires updating of trajectories for every speed update of every block. Furthermore, the same procedure may be repeated many times even though there may not be queries against the updated trajectories. A naive alternative to SUTR is to update trajectories only when answering a query. We call this approach *Query Triggered Revision (QTR)*. For range queries, however, QTR creates a performance problem when the database is large or the query rate is high, since it requires updating of trajectories for every query.

In this paper we propose an approach which avoids any update to the database when answering a range query. This approach follows a filter-refinement paradigm. In the filter stage, the query is relaxed by extending the queried time interval, such that the answer set retrieved by the relaxed query contains all the answers that would have been given by QTR. Then in the refinement stage each retrieved trajectory is revised in main memory and evaluated against the original query. We call this approach *QTR with Query Relaxation*, or *QTR+QR*. We will discuss how the queried time interval is extended in QTR+QR, and will prove that QTR+QR is "correct" in the sense that it returns the identical answer set as QTR. We will also compare it with QTR and SUTR in terms of the extra computational cost introduced by travel-speed prediction.

The rest of this paper is organized as follows. In section

[*]Research supported by NSF Grants ITR-0086144, NSF-0209190, CCR-0070738, and EIA-0000516.

2, we discuss how a time-series prediction method can be used to predict travel-speeds of a road segment in real time. In section 3 we describe the SUTR algorithm and compare it with query processing without travel-speed prediction. In section 4 we describe QTR and QTR+QR, and provide a theoretical and experimental analysis. Finally, the conclusion of this paper is given in Section 5.

2. REAL-TIME TRAVEL-SPEED PREDICTION

We consider only the road segments (namely city blocks) for which travel-speeds are updated periodically. For example, these speed updates may be generated by sensors deployed on the blocks that compute the average speed of the vehicles passing by. We call such blocks *dynamic blocks*. The other blocks are static, i.e. we don't have any information on their real-time speeds. Currently, only highway blocks [1] are dynamic, and blocks of any other type are static.

For the purpose of real-time prediction, for each dynamic block, the system maintains in main memory a history window that stores the latest h speed updates for that block. Let T be the period of speed updating, namely a speed data point is generated every T time units. Then, for each block the history window is $a_1, a_2, ..., a_{h-1}, a_h$, where a_h represents the average travel-speed during the last period of T time units, a_{h-1} represents the average travel-speed during the second last period of T time units, and so on. Future speeds are predicted based on the history window using the Exponential Smoothing method [1]. Specifically, the one-step-ahead speed, namely the average travel-speed for the next T time units, is predicted to be

$$a_{h+1} = \alpha \sum_{k=0}^{h-1} (1-\alpha)^k a_{h-k} + (1-\alpha) \frac{\sum_{k=0}^{h-1} a_{h-k}}{h}$$

where $0 \leq \alpha \leq 1$ is the *smoothing constant*. a_{h+1} is then used to generate the two-step-ahead speed a_{h+2} as if a_{h+1} is a real value, and so on. For selection of the smoothing constant α, a widely used technique [1] is to carry out a sequence of trials on a set of actual historical data using several different values for the smoothing constant, and then to select the value of α that minimizes the sum of squared errors. We adopt this technique in this paper.

In order to keep the predictions realistic, each dynamic block is associated with a maximum speed. If a predicted speed is higher than the maximum speed, it is revised down to the maximum speed. This maximum speed is determined by the speed limit on the street, although may be taken to be slightly higher to account for the fact that vehicles sometimes exceed the speed limit.

3. APPLICATION ON SPATIO-TEMPORAL QUERY PROCESSING

In a moving object database, the motion plan of an object is represented by a trajectory. These trajectories can be constructed based on the average speeds of each road segment. When a query regarding future locations of moving objects is received, trajectories are used for answering. We call this procedure *Query Processing with No Travel-speed Prediction*, or QPNTP. In this section we propose to improve this procedure by incorporation of travel-speed prediction. The rest of the section is organized as follows. In subsection 3.1 we define the trajectory model; in subsection 3.2 we define the spatio-temporal queries and briefly discuss their processing without travel-speed prediction; in subsection 3.3 we introduce query processing with travel-speed prediction; and in subsection 3.4 we compare by experiments the accuracy of query processing with and without prediction.

[1]A highway block is a highway segment between two adjacent exits.

3.1 Representing and Constructing Trajectories

The trajectory model in this subsection is taken from [2].

Definition 1: A trajectory of a moving object is a polyline in three-dimensional space (two-dimensional geography, plus time), represented as a sequence of points (x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n) $(t_1 < t_2 < ... < t_n)$.
A trajectory defines the location of a moving object as an implicit function of time. The object is at (x_i, y_i) at time t_i, and during each time interval $[t_i, t_{i+1}]$, the object moves along a straight line from (x_i, y_i) to (x_{i+1}, y_{i+1}), and at a constant speed given by $v_i = \frac{\sqrt{(x_{i+1}-x_i)^2+(y_{i+1}-y_i)^2}}{t_{i+1}-t_i}$. Thus,

Definition 2: Given a trajectory Tr, the expected location of (the object on) Tr at a point in time t between t_i and t_{i+1} $(1 \leq i < n)$ is obtained by a linear interpolation between (x_i, y_i) and (x_{i+1}, y_{i+1}).

Definition 3: Given a trajectory Tr, its projection on the X-Y plane is called the route of Tr.

Definition 4: A map is a graph, represented as a relation where each tuple corresponds to a block with the following attributes:
– *Polyline:* The 2D coordinates of the shape of the block as a polygonal line segment. Polyline gives the sequence of the endpoints: $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$.
– *Normal Speed:* The normal travel-speed on the block in minutes.
– *Maximum Speed:* The speed limit of the block.

Such maps are provided by, among the others, GDT (www.geographic.com). An intersection of two streets is the endpoint of the four block-polylines. Thus, each map is a graph, with the tuples representing edges of the graph.

A trajectory that represents the motion plan of an object O is constructed as follows. The route of O is specified by giving the source and the destination that O is going to visit. An external routine, available in most Geographic Information Systems, which we assume is given a priori, computes the shortest cost (distance or travel-time) path from the source to the destination in the map graph. This path, denoted $P(O)$, is given as a sequence of blocks (edges), i.e. tuples of the map. Since $P(O)$ is a path in the map graph, the endpoint of one block polyline is the beginning point of the next block polyline. Thus, the whole route represented by $P(O)$ is a polyline denoted by $L(O)$. Given that the trip has a starting time, we compute the trajectory by computing for each straight line segment on $L(O)$, the time at which the object O will arrive to the point at the end of the segment.

3.2 Spatio-temporal Queries and Their Processing without Travel-speed Prediction

We consider two categories of spatio-temporal queries. The first category are *point queries* that pertain to the motion of a single specified object. Two example operators for point queries are as follows.

$Where\_At(Tr, t)$ - returns the expected location of trajectory Tr at time t.

$When\_At(Tr, l)$ - returns the time(s) at which the object on Tr is at expected location l. The answer may be a set of times if the moving object passes through a certain location more than once.

The second category are *range queries* that retrieve the trajectories that intersect a given region during a given time interval. In this paper we consider the following operator for range queries.

$Sometime\_Inside(Tr, R, t_1, t_2)$ - is *true* for a trajectory Tr and a region R iff there exists a time $t \in [t_1, t_2]$ such that the expected location of Tr at t is inside the region R.

Let us identify the topological properties which are necessary and sufficient conditions for satisfaction of the above predicate. Taking time as the third dimension, the region R along with the queried time interval $[t_1, t_2]$ can be represented as a prism P_R in 3D space: $P_R = \{(x, y, t)|(x, y) \in R \wedge t_1 \leq t \leq t_2\}$. P_R is called the *query-prism*.

$Sometime\_Inside(Tr, R, t_1, t_2)$ is true iff Tr intersects the query-prism P_R.

For the purpose of query processing, we assume an available 3D indexing scheme in the underlying DBMS.

In the rest of this paper we assume that $[t_1, t_2]$ is in the future. Otherwise query processing is not affected by travel-speed prediction. Observe that when $[t_1, t_2]$ is in the future, answers to queries are all estimated, or expected in the future, based on current information. The objective of prediction is to improve the estimation.

3.3 Query Processing with Travel-speed Prediction

When a speed update is received for a dynamic block B, it is added into the history window. Future speeds are predicted for B for the next L time units (L is a system parameter). For any time after $now + L$, the normal speed of B provided by the map is taken to be the speed at that time. The predicted speeds are then used to update all the trajectories that pass B within the next L time units. Since trajectories are revised as a result of speed updates, this approach is called *Speed Update Triggered Revision*, or *SUTR*.

3.4 Comparison of SUTR and QPNTP

In this subsection we compare SUTR and Query Processing with No Travel-speed Prediction in terms of how accurately they answer queries.

3.4.1 Data Settings

The time-series data used in our simulations are travel-speed time-series of the northbound Edens Expressway in the Chicago area, which are published and periodically updated on the web site www.ai.uic.edu/GCM/Edens.html. These data are generated by loop detectors on each of the 36 northbound blocks in the expressway. Each detector averages all the speeds of the cars going over it and sends over this value every five minutes. Thus the speed update period is 5 minutes. We collected history time series for each of the 36 blocks for 20 days, resulting in 5760 data points for each block (20 days × 24 hours × 60 minuntes / 5 minutes). For each block we used 10 days data to get the best value of the smoothing constant, and used another 10 days to conduct simulations.

3.4.2 Simulation Method

We randomly generated 5000 trajectories along the 36 blocks. Each trajectory is constructed based on 3 parameters: a start block (sb), a destination block (db), and an initial time (t_0). With these parameters, a moving object travels along the expressway from the start block to the destination block, starting at the initial time. The start block is randomly selected from interval $[1, 35]$. Once the start block is determined, we pick up a random value from the interval $[sb + 1, 36]$ as the destination block. The initial time is assigned a random value from 0 to 2880×5 minutes where 2880 is the number of 5 minute intervals within 10 days.

Each simulation run is executed as follows. A range query is issued at the beginning of each 5 minute interval within 10 days, which gives us 2880 queries. Each query has the form of $Sometime\_Inside(Tr, b, t+f, t+f+5)$, which is issued at time t and retrieves the moving objects that are expected to be inside block b sometime between $t + f$ and $t + f + 5$. For each query, b is randomly chosen from $[1, 36]$. The lead time f is fixed during a simulation run, and it ranges from 5 to 60 minutes with the step size of 5 minutes. When a query is issued, since from the data we know the actual speeds of each block in each time interval, we know where a simulated object will be from $t + f$ until $t + f + 5$. Thus we know the correct answer set of each query. On the other hand, a speed update is generated for each block for every 5 minute interval, and SUTR is executed upon each speed update. Exponential smoothing is used for travel-speed prediction in SUTR. L is set to be 1 hour, namely the futures speeds are predicted up to 1 hour ahead.

The accuracy of the answer set to a query is measured by the sum of recall and precision. Recall means the portion of retrieved objects out of the correct answer set, whereas precision is that of correctly returned ones out of those retrieved. Let S_{return} be the set of retrieved trajectories by a query, $S_{correct}$ be the set of correct answers. When the number of elements in the set S is denoted by $|S|$, the recall and the precision are defined as follows:

$$recall = \frac{|S_{return} \cap S_{correct}|}{|S_{correct}|},\ precision = \frac{|S_{return} \cap S_{correct}|}{|S_{return}|}$$

3.4.3 Simulation Results

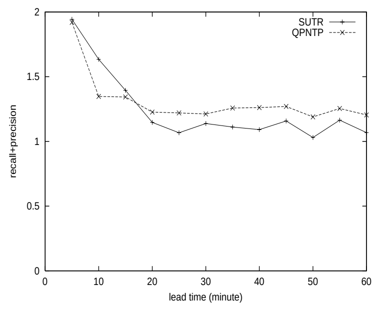

Figure 1: Sum of precision and recall as a function of the lead time

Figure 1 shows the sum of recall and precision as a function of the lead time (f). It can be seen that when the lead time is smaller than 15 minutes, the accuracy of SUTR is higher than that of QPNTP, and the reverse is true afterwards. This indicates that travel-speed prediction should be performed for no more than 15 minutes ahead, for the exponential smoothing method.

4. QUERY TRIGGERED REVISION WITH QUERY RELAXATION

4.1 Motivation

SUTR requires updating of trajectories for every speed update of every dynamic block. This incurs a performance problem when the speed update rate is high or the size of the highway network is large. Furthermore, the same procedure may be repeated many times even though there may not be queries against the updated trajectories.

A natural solution to this problem is that, instead of revising trajectories for each speed update, revise them only when answering a query. We call this approach *Query Triggered Revision*, or *QTR*. For point queries, QTR works as follows. When a point query is received at time t, the trajectory of the queried object, denoted Tr, is retrieved from the database. The dynamic blocks that are traveled after t on the route of Tr are identified. The future speeds for these blocks are predicted. Tr is then revised based on the predicted speeds. Finally, the revised Tr is used to answer the query.

For range queries, QTR works as follows. When a range query $Sometime\_Inside(Tr, R, t_1, t_2)$ is received at time t, the set of the trajectories that intersect R after t is identified. This can be done by running through the index tree using a prism whose base is R and its height is $t_{max} - t$ where t_{max} is the maximum among the end times of all the trajectories. Denote this set C. The dynamic blocks that are traveled after t on the routes of the trajectories in C are found, and the future speeds for them are predicted. The trajectories in C are then updated with the predicted speeds. Finally the query is evaluated on each updated trajectory.

Observe that for range queries, QTR requires updates to the database trajectories every time a query is received. This incurs a performance problem when the database is large or the query rate is high. Further observe that in both SUTR and QTR, the database trajectories change as a result of travel-speed predictions.

In this section we propose an approach that avoids any update to the database when answering a range query. This approach is called *QTR with Query Relaxation*, or *QTR+QR*. QTR+QR improves QTR by introducing a filter stage. In this stage, the original query is relaxed by extending the query interval, such that the trajectories retrieved by the relaxed query include all the answers that would have been returned by QTR. Then in the refinement stage, each retrieved (filtered) trajectory is revised in main memory and evaluated against the original query.

4.2 Description of QTR+QR

When a range query $Sometime\_Inside(Tr, R, t_1, t_2)$ is received, the system follows a filter-refinement strategy to process the query. In the filter stage, the future speeds for each dynamic block for the next L time units are predicted (recall that L is the maximum lead time for travel-speed prediction). The query interval $[t_1, t_2]$ is then extended based on how the predicted speeds will affect the trajectories. For example, if the predicted speeds are expected to cause delays for all the objects, then the start time of the query interval is extended to $t_1 - L$. In the 3D space, this interval extension is reflected by a down-shift of the base of the original query-prism (see Figure 2). The relaxed query is used to retrieve a set of candidate trajectories. In the refinement strategy, each candidate trajectory is revised with the predicted speeds and evaluated against the original query-prism. Observe that saving the revised trajectories back to the database would be the natural, and probably naive way of doing so. However, it will result in many updates to trajectories that may never be queried.

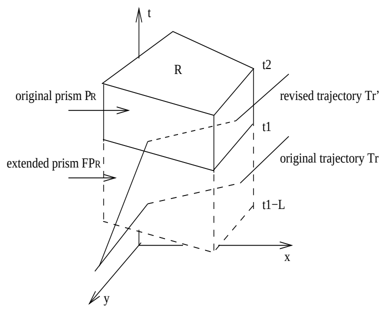

Figure 2: Extension of the query-prism

Now we formalize the procedure of QTR+QR.

Query Triggered Revision with Query Relaxation

Input: Original query $Q = Sometime\_Inside(Tr, R, t_1, t_2)$. The normal speed $\overline{v}_i$ and the maximum speed m_i of each dynamic block i. The latest h speed updates for each block.

Step 1: For each dynamic block i, predict the speed time series S_i for the next L time units. If a predicted speed is higher than m_i, revise that speed to m_i. Let v^i_{high} be the highest speed in S_i and v^i_{low} the lowest one.

Step 2: Generate the relaxed query RQ as follows.
1. If for every block i, $v^i_{high} < \overline{v}_i$, then $RQ = Sometime\_Inside(Tr, R, t_1 - L, t_2)$;
2. If for every i, $v^i_{low} > \overline{v}_i$, then $RQ = Sometime\_Inside(Tr, R, t_1, t_2 + L)$;
3. Otherwise, $RQ = Sometime\_Inside(Tr, R, t_1 - L, t_2 + L)$.

Step 3: Use RQ to query the database, obtaining the candidate set C.

Step 4: For each trajectory Tr in C, revise Tr with the predicted speeds. Let C' be the set of the revised trajectories.

Step 5: Refine by evaluating Q on each trajectory in C'.

4.3 Correctness Proof of QTR+QR

In this subsection we prove the correctness of QTR+QR. We show that, under some very realistic assumption, QTR+QR returns the same answer set as QTR. In the following analysis, we use Tr to denote an original trajectory in the database and Tr' the one revised with the predicted travel-speeds. Denote by Q a range query and by RQ its relaxed query.
Lemma 1: Assume that for each dynamic block the maximum speed is at most twice the normal speed. If Tr' satisfies Q then Tr satisfies RQ.

Proof sketch: Let t be the time when the query is issued. Let t_p be the time when the object is expected to arrive at a location p according to Tr, and t'_p be that according to Tr' (p is a point on the route of Tr). Consider the difference between t_p and t'_p. There are three cases.

(1) $t'_p > t_p$, i.e. the revised trajectory causes a delay at p. The maximum possible delay is L, and this delay occurs when the predicted travel-speed during the time interval $[t, t+L]$ is 0 and thus the object is expected to be stationary from t to $t+L$ according to Tr';

(2) $t'_p < t_p$, i.e. the revised trajectory causes an advance at p. The maximum possible advance is L. This advance occurs when the speed of Tr' is the maximum speed for each block on the route of Tr' from t to $t+L$ (since that maximum speed is twice than the normal one).

(3) $t'_p = t_p$.

In any of the above three cases, the difference between t_p and t'_p is no more than L. Thus if a point a' of Tr' is inside the original query-prism Q, then the point a of Tr corresponding to the location of a' is inside the extended query-prism RQ. $\Box$

Lemma 1 suggests that QTR+QR guarantees no "false dismissals" during the filter stage, i.e. the answer set returned by the relaxed query is a superset of the answer set of QTR. The assumption about the ratio between the normal speed and the maximum speed is realistic. For example, the normal speed of all the Edens highway blocks is 45 miles/hour. Lemma 1 holds as long as the maximum speed does not exceed 90 miles/hour.

Theorem 1: If for each dynamic block the maximum speed is at most twice the normal speed, then the answer set given by the QTR+QR method for Q is identical to that given by the QTR method.

4.4 Comparison of QTR+QR, QTR, and SUTR

In this subsection we compare QTR+QR with QTR and SUTR in terms of the number of trajectory revisions for each one. We do this for the following reason. The cost of trajectory maintenance and query processing can be decomposed into two components. The first component is the cost of running the query-prism through the index tree. This cost is a constant for a query, regardless of the query processing method used. The second component is how many trajectories are evaluated and updated individually, which depends on the method used.

The simulation setup is similar to that in section 3. The trajectory database with 5000 trajectories resides in the main memory during all the simulations. Let the query rate be qr queries per minute. Each simulation run is executed as follows. qr queries are generated for every minute within 10 days, which gives us $qr \times 60 \times 24 \times 10$ queries. Each query has the form of $Sometime\_Inside(Tr, b, t+f, t+f+5)$. t is the time when the query is issued. b is randomly chosen from $[1, 36]$, and f is randomly chosen from 5, 10, ..., 60 minutes. L is set to be 15 minutes for all the three approaches.

We experimented with qr ranging from 1 to 20. Figure 3 shows the total number of trajectory revisions as a function of the query rate. Observe that the numbers for QTR and QTR+QR increase as the query rate increases, whereas that for SUTR does not change. This is because SUTR revises trajectories only as a result of speed updates but not as a result of queries; whereas QTR and QTR+QR revise trajectories only as a result of queries. Further observe that

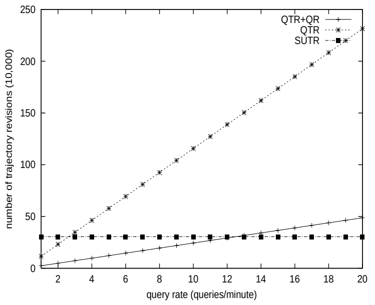

Figure 3: Number of trajectory revisions as a function of the query rate

QTR+QR is always better than QTR and the difference between them increases as the query rate increases. This verifies that QTR+QR significantly reduced trajectory revisions by excluding a lot of trajectories in the filter stage. Actually the slope of $QTR + QR$ is much flatter than that of QTR, indicating that $QTR+QR$ is much less sensitive to the query rate than QTR. Finally, QTR+QR is better than SUTR when the query rate is lower than 12 queries/minute, and after that SUTR is better. The threshold 12 is a function of the number of trajectories in the database (5000 in our case), and will increase as the number of trajectories increases.

5. CONCLUSION

We examined travel-speed prediction for query processing. We proposed three methods of applying travel-speed prediction, namely SUTR, QTR, and QTR+QR, and analyzed them theoretically and experimentally. The conclusions are (i) for the short term future (15 minutes or less), query processing with travel-speed prediction provides more accurate answers than without travel-speed prediction; (ii) QTR+QR is equivalent to QTR (i.e. it provides the same answer set), but with a much lower number of trajectory revisions; (iii) QTR+QR and SUTR are suitable for different situations depending on the query rate.

The accuracy of query processing with travel-speed prediction may be improved by using a more sophisticated travel-speed prediction method. For example, multivariate models [3] take into considerations the interactions between the traffic of neighboring blocks. These models proved to be superior to univariate methods (such as the one used in this paper) as far as short term forecasting is concerned, and they are the subject of future work.

6. REFERENCES

[1] D. Montgomery, L. Johnson, and J. Gardiner. *Forecasting and Time Series Analysis*. McGraw-Hill, 1990.

[2] G. Trajcevski, O. Wolfson, F. Zhang, and S. Chamberlain. Geometry of uncertainty in moving objects databases. In *Proc. of the 8th Int. Conf. on Extending Database Technology (EDBT 2002)*, pages 233–250, March 2002.

[3] A. Stathopoulos and G. Karlaftis. A multivariate state-space approach for urban traffic flow modeling and prediction. In *81th Annual Transportation Research Board Meeting*, 2002.