Context-Aware Optimization of Continuous Range Queries Maintenance for Trajectories

Goce Trajcevski Hui Ding Peter Scheuermann*

goce,hdi117,peters@ece.northwestern.edu

Department of Electrical and Computer Engineering
Northwestern University
Evanston, Il 60208

ABSTRACT

This work addresses the problem of efficient *maintenance* of the (correct) answers to the continuous spatio-temporal range queries in Moving Objects Databases (MOD), which represent the objects' motion as *trajectories*. Specifically, we consider the settings of optimizing the *response time* of the system when the queries need to be brought up-to-date as a result of *bulk update* to the trajectories in the MOD. Such updates occur when an *abnormality* occurs in some context dimension (e.g., road accident; fire) that affects many trajectories in a given region. However, the updates of those trajectories may affect the correctness of the answers to queries which pertain to regions that are not spatially close to the region where the abnormality occurred, and are interested in some future-time with respect to the time of the occurrence of that abnormality.

Categories and Subject Descriptors: H.2.4 [Database Management]: Systems: *query processing*

General Terms: Algorithms

Keywords: Moving Objects Databases, Triggers

1. INTRODUCTION

Location management [11] is an enabling technology for many novel classes of applications and the management of the transient *(location,time)* information of a large number of mobile entities has been a subject of many research efforts in the field called *Moving Objects Databases* (MOD). Various categories of problems of interest to MOD have been investigated – *access methods (indexing), modelling, query processing* and a relatively recent collection with many relevant references is presented in [6].

Several research prototypes have been implemented (e.g., [2, 7]) dedicated to various categories of problems of interest to MOD. However, on the commercial side, there have been very few database vendors that have enabled MOD-like capabilities in their products. A particularly appealing commercial Object-Relational Database Management System (ORDBMS) is Oracle[1] 9i [9] which, on top the reliability of a stable and mature technology, also offers:
1. *PL/SQL* as an environment for implementing User-Defined Functions (UDF) and specifying User-Defined Types (UDT);
2. *Oracle Spatial* which provides the `Linear Referencing System` (LRS) and the `SDO_GEOMETRY` type for specifying and managing (querying) spatial objects, as well as the ability for using spatial indexing;
3. *Triggers* as declarative means to specify a *reactive* behavior in response to modifications to the MOD.

When it comes to representing the *motion* of the objects, there are three distinct models which are, in a sense, "attractor centers" along the spectra: 1. Sequence of *(location, time)* updates periodically reported to the MOD (e.g., obtained by a GPS device); 2. Sequence of *(location, time, velocity)* updates, which are reported only when an object deviates from the expected motion according to the previous update; 3. *trajectory*, which represents the future motion of an object and is obtained based on the points that an object intends to visit and some extra information (map and speed patterns). In this work we assume that the motion of each object is represented as a trajectory in the MOD. This is the case for many entities in practical settings, e.g., public transportation vehicles, police patrol cars, delivery trucks, individuals going back-and-forth between home and work, etc. One of the peculiarities of the MOD queries is that they are not only *instantaneous*, but can also be *continuous*, which is, pertaining to the future, and may have to be re-evaluated upon modifications to the MOD [14]. Regardless of the motion model adopted, the MOD will have to *react* to the changes, in order to maintain the correctness of the (pending) continuous queries.

1.1 Motivational Scenario

To get a better intuitive idea about the main aspects of our work, observe the scenario depicted in Figure 1. It illustrates three spatio-temporal range queries **Q1**, **Q2** and **Q3** (solid prisms) and six trajectories **Tr1**, ..., **Tr6**, rep-

*Research partially supported by NSF grant IIS-0325144

MobiDE'05, June 12, 2005, Baltimore, Maryland, USA.

[1]We would like to point out that our approach can be implemented on any ORDBMS that features triggers whose semantics conforms with the SQL99 standard.

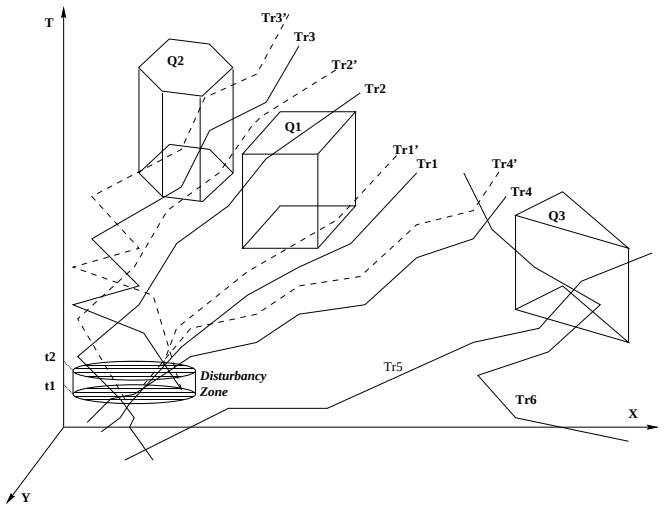

Figure 1: Trajectories, Updates and Continuous Range Queries

resenting the expected future motion plans of six objects (solid polylines). After the queries have been posed, a traffic abnormality occurs at some time t_1, which affects the distribution of the speed patterns on the segments in the particular *Disturbance Zone*. The abnormality persists until some time t_2 and its effect is that any object moving through the Disturbance Zone will have to slow down for a certain percentage of its expected speed. Thus, the future-portions of the *affected* trajectories will have to be updated (**Tr1'**, **Tr2'**, **Tr3'** and **Tr4'**) as illustrated with dashed lines. This, in turn, affects the answers to the pending continuous queries. For example, **Tr2'** is no longer part of the answer to the query **Q1**, but instead becomes a part of the answer to **Q2**, which was not the case for **Tr2**, prior to the traffic abnormality. Clearly, upon updating the trajectories, it is necessary to re-evaluate the pending queries, however, there are subtleties which are the main motivations for this work: 1. The query **Q3** need not be re-evaluated because none of the trajectories in its answer is affected by the given traffic abnormality *and* none of the trajectories affected by the abnormality has any impact on **Q3**; 2. When re-evaluating **Q1** and **Q2**, we need not take **Tr5** and **Tr6** into consideration.

Our goal in this paper is to optimize the time that it takes to re-evaluate a set of pending queries in a MOD when a set of trajectories is being updated. Clearly, this is a form of a *reactive* behavior triggered by changes in spatio-temporal context dimensions and we show how the dependencies among the values in the spatio-temporal dimensions and the various system-level context dimensions can be exploited in order to optimize the *response time* when re-evaluating a set of continuous range queries. Our main contribution are:

• We investigate the impact of the values of various *semantic dimensions* (c.f. [10]), as declaratively specified for each individual trigger, on the penalties due to the *context-switching*.

• We utilize the spatio-temporal correlation between the pending queries and (updated) trajectories to: 1. limit the search space and 2. to generate an ordering among the triggers' execution that will improve the response time.

• We have completely implemented our system[2] on top of an existing ORDBMS – Oracle 9i using triggers (whose processing was subject to optimization) to achieve the reactive behavior, and we provide experimental results which demonstrate the validity of our approach.

The rest of this paper is structured as follows. In Section 2 we introduce the notation and set up the foundation for presenting our main results in Section 3. Section 4 presents our experimental observations and in Section 5 we position our work with respect to the related literature and outline directions for future work.

2. PRELIMINARIES AND SYSTEM ARCHITECTURE

In this section we introduce the terminology and the main concepts used in the rest of the paper and we describe the basic components of our system.

The spatio-temporal nature of a given moving object, is represented using a *trajectory* [18], which is a piece-wise linear function $f : T \rightarrow (x, y)$, represented as a sequence of points (x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n) $(t_1 < t_2 < ... < t_n)$. For a given a trajectory Tr, its projection on the XY plane is called the *route* of Tr. This representation entails that the object is at (x_i, y_i) at time t_i, and during each segment $[t_i, t_{i+1}]$, the object moves along a straight line from (x_i, y_i) to (x_{i+1}, y_{i+1}), and at a constant speed. The *expected location* of the object at any time $t \in [t_i, t_{i+1}]$ $(1 \leq i < n)$ is obtained by a linear interpolation between (x_i, y_i) and (x_{i+1}, y_{i+1}).

Relative to *now*, a trajectory can represent both the *past* and the *future* motion of objects. The future part of a trajectory corresponds to a *motion plan* of the moving object and we construct[3] it using an *electronic map* like, for example, the ones provided by Geographic Data Technology *(www.geographic.com)*. The map is, essentially, a graph, represented as a relation where each tuple corresponds to a block with attributes such as:

– Polyline: *Each block is a polygonal line segment. Polyline gives the sequence of the endpoints:* (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)

– Length: *Length of the block.*

– Fid: *The block* id *number.*

– Drive_Time: *Typical drive time from one end of the block to the other.*

An important observation is that the Drive_Time attribute, commonly provided by the electronic maps, is "static", in the sense that it does not consider variations of the traffic (e.g. the speed in a given block is 35 mph during mid-day hours, and 15 mph during "rush" hours). Similarly to [18], we assume *Speed_Profile* attribute for each block which captures the distribution of the traffic patterns during a given time-period (e.g., a day) and this is used in constructing the trajectory. We assume that a given moving object transmits its *start_point* and *start_time*, and the *destination_point* (possibly, a sequence of other "to-be-visited" points too), which are used as input to the *time-dependent* (A^* extension) variant of the Dijkstra's shortest path algorithm, where the cost

[2]The source code of the PL/SQL routines that we wrote, as well as the data set used in our experiments is available from *http://www.ece.northwestern.edu/peters/ContextMOD*

[3]See [18] for a more detailed description of the trajectories' construction.

of an edge in a graph depends on the start time to travel along that edge. Using this, we generate the shortest (in travel-time or distance) path and for each straight line segment, we compute the object's arrival time at its end-points. The *past* trajectory of a given object can be constructed using a set of 3D points (x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n) which transmitted by the moving object periodically, say using an on-board GPS to detect location at given time points. The points are "snapped" on the map then connected with the time-dependent shortest path.

As we mentioned in the Introduction, throughout this work we will focus on the spatio-temporal *range* queries of the form:

Qi: *"Retrieve all the objects that will be inside the region R_i between the time t_{bi} and t_{ei}"*

R_i denotes the (spatial) *region* of interest and t_{bi} (resp. t_{ei}) denote the *begin* (resp. *end*) time-values of interest. One may observe that there can be different variations of the query **Qi** depending whether the object is inside R_i *sometime* or *always* (or even certain percentage of the time) between t_{bi} and t_{ei} [18], however, this is not relevant for the problems addressed in this paper and, without loss of generality, we assume the *sometime* semantics. We are only considering queries that pertain to the *future* of the MOD – in other words, we assume that the query **Qi** was posed at some time $t_{pi} \leq t_{bi}$. A detailed classification of categories of spatio-temporal queries and the impact of the (semantics of the) relevant elements of their syntax is given in [16] and, due to lack of space we omit it here. In the rest of this work, we will use **A_Qi** to denote the answer-set of a particular query **Qi**.

We assume that, once **Qi** has been processed, the MOD transmits the answer to the user that posed the queries. However, the MOD needs to keep on monitoring **A_Qi**, at least until the time t_{ei}, because its value may change. There are several sources that could change **A_Qi**: – *insertion* of new trajectories in MOD; – *deletion* of existing trajectories from MOD; – *update* of a trajectory, specifically requested by a given user. However, one particularly interesting source, which is the topic of this work, is when unexpected abnormalities (e.g., an accident, a flood) occur. They affect a given region and may change the (expected) values of some of the parameters used in some trajectories' construction. Specifically, the *Speed_Profile* attributes along some segments may be impacted which, in turn, requires that the *future-motion* of the associated trajectories be reconstructed. Consequently, some of the answers to the pending continuous queries may need to be re-evaluated and the *optimization* of this process is our main goal.

Figure 2 illustrates the main components of our system:

• *Moving Objects Table* (MOT) which stores the trajectories (plus some other static attributes like `name`, `model`, etc.) It is defined as

```
CREATE TABLE MOT
 traj_id number primary key,
 traj_name varchar2(30),
 traj_shape MDSYS.SDO_GEOMETRY,
```

The *traj_shape* attribute is the spatio-temporal attribute representing the trajectory of the moving object. It is defined as of type SDO_GEOMETRY, which is a pre-defined type in *Oracle Spatial*. In order to speed up the *filtering* stage for processing queries (and updates), the `traj_shape` attribute is indexed (`INDEXTYPE IS MDSYS.SPATIAL_INDEX;`).

• *Traffic Abnormalities Table* (TAT) which stores the information about the disturbances which affect some of the parameters used in trajectories construction. The main attributes of this table are:

```
Disturbancy_Zone MDSYS.SDO_GEOMETRY,
Duration timestamp,
Disturbancy_Type UDT
```

The `Disturbancy_Type` is a User-Defined Type that contains the information specifying the effect(s) that a particular disturbance has on the road segments in the region of its `Disturbancy_Zone`. Essentially, it describes the type of the disturbance (e.g., road-work, accident) and its impact on the *Speed Profile* attribute, in terms of the relative slowdown effect on the road segments. Typically, a trajectory entering a road segment in the *Distrubancy Zone* will have a short period of *deceleration*; followed by a period of motion with a constant speed, but much slower than normal (the expected value for that period); followed by an *acceleration* on the road segment exiting the *Disturbance Zone*, until the motion reaches the expected value of the *Speed Profile*. The details of incorporating these effects in the process of *updating* future portions of the trajectories affected by a particular abnormality are presented in [19].

• *Queries Table* (QT) stores the information pertaining to (pending) queries posed to the MOD, with attributes:

```
Query_ID number,
  Region  MDSYS.SDO_GEOMETRY,
  begin_time timestamp,
  end_time timestamp
  Current_Answer UDT
```

Viewed in 3D (c.f. Figure 1), the query occupies a volume of a prism with base at `Region` located at the horizontal plane `begin_time`, and with a height `end_time - begin_time`. The `Current_Answer` attribute is essentially a UDT representing the list of the trajectories in the answer-set for a particular query.

Once the system receives a new query request from a given user, its interface extracts the elements of its syntax necessary for its maintenance (e.g., the t_{ei} specifies the time after which the query no longer needs to be monitored and can be purged from the system). The basic paradigm for maintaining the answer to a given continuous query can be described as follows:

When a new query Q_{new} is posed:

1. *Process Q_{new} and get its answer A_Q_{new};*
2. *Transmit the answer to the user;*
3. *Set up a trigger TR_Q_{new} of the form:*

   `ON UPDATE TO MOT`
   `IF` A_Q_{new} `Affected`
   `Update` A_Q_{new}

In terms of the actual execution model, the step of checking if A_Q_{new} was affected requires evaluating the query on the modified trajectory(ies). On the other hand, for the update of A_Q_{new} it suffices to change *current answer* attribute of Q_{new} in QT.

The necessary components of a system that will ensure *correctness* of the reactive behavior under this paradigm were presented in [16] and an implementation for maintaining a particular type of continuous spatio-temporal queries – (*Within Distance*) – was presented in [17], however, in those works we were not concerned with the *optimization* of the reactive behavior. In this paper, we focus on range queries, and we assume that the sources for the reactive be-

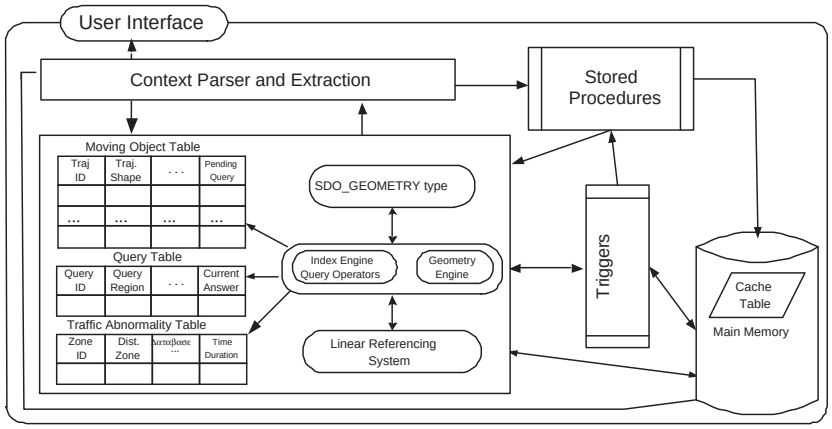

Figure 2: System Architecture

havior are modifications to the TAT that enable a trigger **TR_TAT** whose high-level description is:

```
ON INSERT/UPDATE to TAT
IF MOT.traj_shape affected
UPDATE MOT.traj_shape
```

We would like to point out that the optimization of the *query processing* for the purpose of initial/instantaneous generation of the answer is *not* the topic of this work. In our settings, given the trajectory model that we assume, the algorithms presented in [3] can be straightforwardly adopted and implemented in PL/SQL. Our main goal is to optimize the *processing time* spent from the moment a particular abnormality is presented to the MOD up to the point that all the pending continuous queries' answer-sets are brought up-to-date. We focus on orchestrating the execution of the respective triggers, based on intelligent usage of the dependencies among various context/semantic dimensions as main means towards our goal. This makes the results of our approach valid regardless of which particular query processing strategies are used for instantaneous evaluation.

3. THE LEVELS OF CONTEXT-AWARENESS

We address now the various aspects of optimizing the maintenance of a set of pending continuous range queries in a MOD. Specifically, we target the minimization of the computational overhead at several *semantic* (*context*) dimensions, both at the *system level* and the *spatio-temporal relationship level*

3.1 System-Level Context Awareness

One of the sources of significant overhead in the re-evaluation of pending continuous range queries is the penalty involved in *context-switching*. Recall (c.f. Section 2) that upon receiving a particular query, say **Qi**, our system, besides initiating the calculation of its answer, automatically sets up a trigger **TR_Qi** which will react to the updates to the MOT table. In our setting, this can only occur when the TAT table is updated due to traffic abnormalities which, in turn, "awakes" the **TR_TAT** trigger, whose action part generates the enabling event of **TR_Qi**. The syntax of **TR_Qi** is:

```
1. CREATE TRIGGER TR_Qi
2. on UPDATE to MOT
3. if A_Qi affected
4. Update A_Qi
```

and in the sequel we analyze the impact that some of the options available for specifying **TR_Qi** have on the response time.

• *Set vs. Instance Level*: One of the semantic dimensions that may have different values in a particular trigger, as specified in the SQL99 [1] standard (as well as being implemented in the Oracle 9i [9]) is the response to modifications of a particular table in an *instance (tuple)*-oriented and *set*-oriented manner [10]. To achieve the behavior corresponding to each of these options, one needs to use (typically, after line 2.) in the specification of **TR_Qi** the statement either `FOR EACH ROW`, or `FOR EACH STATEMENT`, respectively.

If the executional semantics is `FOR EACH ROW` then, whenever a particular trajectory Tr_j is updated, the corresponding instance of **TR_Qi** will evaluate it against **Qi**, in order to check if the modifications to Tr_j affect the answer-set **A_Qi**. If so, **A_Qi** will updated accordingly. This behavioral cycle will be repeated for the subsequent trajectory, say Tr_k, that was affected by the update to the TAT. However, at each cycle, we have (at least) three major *context-switchings* that the (OR)DBMS[4] needs to do, and will be reflected at the Operating System (OS) level: 1. From *update* of the MOT mode to *querying* if **A_Qi** is affected; 2. From *querying* if **A_Qi** is affected to (eventually) *updating* it; 3. From *updating* **A_Qi**, back to the *update* mode for processing the modifications to the Tr_k.

On the other hand, if **TR_Qi** is specified to execute in a `FOR EACH STATEMENT` manner, then *all* the trajectories (like Tr_j and Tr_k) affected by the traffic abnormality will be updated in the MOT first and, subsequently, the ORDBMS will proceed towards evaluation of the condition part of the **TR_Qi** for all the updated rows. Clearly, this behavior, achieves some savings in the *context-switching* overhead which are far from negligible, as illustrated by the experiments in Section 5.

• *Before vs. After Triggers*: We re-iterate that, from the perspective of the user who posed a given query **Qi**, the most important issue is that *answer-set* of the **Qi** be brought to "current" state as soon as possible. Part of that race can be won if one observes that the modifications (updates) of the trajectories that are affected by the traffic abnormality *need*

[4]The specification of the SQL99 standard distinguishes between *statement execution context*, *routine execution context*, and *trigger execution context* [1].

not be *actually completed* in the database table (MOT), in order to bring the queries' answers up-to-date. Once the trajectories that are affected by the update to the TAT are identified and brought in the main memory and their new shapes are calculated, the information needed to re-evaluate **Qi** is already available.

In order to utilize this kind of behavioral optimization for the response time of the pending queries in the MOD, we can utilize some of the options provided by the Oracle 9i [9] (and, again, part of the SQL99 standard [1]): the BEFORE option for triggers' specification. Namely, line 2. of the trigger **TR_Qi** can be specified either as:

`2'. BEFORE UPDATE on MOT` or as:

`2''. AFTER UPDATE on MOT`, which is the default behavior (if the specification is omitted).

Both specifications will generate correct updates of the answer-set of **Qi**, however, as we demonstrate in Section 5, the `BEFORE` option yields much faster response time. Again, this is due to *context-switching* issues because the BEFORE mode avoids an extra retrieval for the set of the updated trajectories that could potentially affect some of the pending queries.

3.2 Intra-Query Relationship and Triggers Selection

In the previous section we discussed the benefits of avoiding some of the *context-switching penalties*, by carefully specifying the syntax of the trigger corresponding to a continuous range query. However, there is another dimension of *context awareness* which, if utilized, can further improve the minimization of the response time for maintaining the answer-sets of the pending continuous queries. This is especially significant when there is more than one query posed to the MOD.

• *Selecting the relevant subset of triggers*: Let us recall query **Q3** from our motivational scenario in Section 1.1). None of the trajectories affected by the given traffic abnormality at the particular disturbance zone impact the correctness of the answer-set **A_Q3**. In order to utilize this kind of intelligent behavior in our setting we propose to do some extra work when a particular query is submitted which,enables our system to behave in an "output sensitive" spirit when processing the respective triggers. Similar ideas have been used in [12, 21], however, the motion model used in these works – a sequence of *(location,time)* updates as the objects are moving – is different from our representation of the trajectories.

The basic idea is to *index the queries* (c.f. [12]) as they are being posed to the system. In Oracle 9i, this is achieved by the statement:

```
CREATE INDEX query_lrs_idx
  ON tbl_query_lrs(query_region)
   INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

Subsequently, after updating the TAT due to a traffic abnormality, as each trajectory is being accordingly modified, we maintain the *Minimum Bounding Box* (MBB) of the updated portions of the trajectories "on the fly". When determining the triggers that need to be fired in order to re-evaluate their respective pending queries, we only choose a subset of them, which consists of the ones whose with the queries' 3D prisms intersect the MBB constructed during the process of updating the trajectories. This will prevent re-evaluation of queries like **Q3** in our motivational scenario.

• *Ordering among the triggers*: Whenever there is a set of triggers enabled by a same event, upon that event's occurrence, there must be some priority criteria for *ordering* among the triggers, in the sense of when a particular trigger will have its condition evaluated (and, eventually, its action executed[5]). Some prototype active database systems actually allowed the programmer to specify a pre-determined criterion as part of the syntax of a trigger (i.e., PRECEDES and FOLLOWS clauses) [10]. However, Oracle 9i orders the triggers by the time-stamps of their creation in the system and does not allow for explicit specifications any ordering criteria by the user(s). To overcome this, we had to write our own PL/SQL procedures that will reflect user's criteria in the actual triggers selection.

Can ordering among the triggers be used to, potentially, eliminate certain computational overheads? The answer is *yes*, due to some subtle issues related to the LRU policy used for swapping the pages between the disk and the main memory. Thus, if two range queries like **Q1** and **Q2**, which correspond to the condition evaluations of two distinct triggers, are "close" in spatio-temporal sense, there is a high chance of per-page(s) overlap of the MOT data used by both of them. Consequently, if there is a third query, like **Q3** *any* of the sequences: **(TR_Q1, TR_Q2, TR_Q3)**, **(TR_Q2, TR_Q1, TR_Q3)**, **(TR_Q3, TR_Q1, TR_Q2)** or **(TR_Q3, TR_Q2, TR_Q1)** of executing the (condition parts of the) corresponding triggers, will have much higher chance of minimizing the swap-overhead as opposed to the sequences: **(TR_Q1, TR_Q3, TR_Q2)** or **(TR_Q2, TR_Q3, TR_Q1)**. In order to achieve this benefit, one should sort the triggers in such a manner that their ordering sequence exploits to the maximum the spatio-temporal proximity of the corresponding queries' parts.

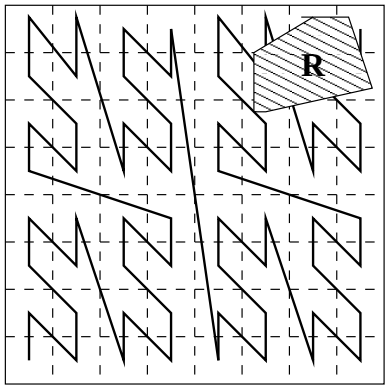

Figure 3: Space-filling Z-curve

Towards this goal, the approach that we took in our system is to use an ordering among the triggers based on space-filling curves [8]. As illustrated in Figure 3, we impose a grid over the entire region of interest and we order the cells according to their position of the sequence based on the Z-curve (sometimes also referred to as Peano curve) [8]. To determine the value of a particular trigger in the precedence

[5]The detailed analysis of the impact of the *coupling* modes [10] of the different parts of a particular trigger (Event, Condition or Action) among themselves and with the transaction that generated its enabling Event is beyond the scope of this paper.

order, we use the cell to which the geometric centroid of the region **R** of the associated query belongs. As a conflict-resolution policy, in case that the centroids of two (or more) query-regions fall in the same cell, we used the *begin-time* parameter of each query to determine the order among them. As our experiments demonstrate, this exploitation of the spatial proximity indeed achieves improvements in the response time.

In summary, the analysis that we conducted in this section has shown a number of tuning options which can improve the response time for maintenance of the spatio-temporal range queries in our settings:

• An index should be maintained for the query regions which can be checked against the MBB of the updated trajectories, in order to eliminate the triggers that need not be executed.

• Triggers should be ordered according to the Z-curve ordering of their corresponding query regions.

• Each individual trigger should be specified to execute in BEFORE and FOR EACH STATEMENT manner.

4. EXPERIMENTAL RESULTS

In this section we assess the effectiveness of our methodology by presenting experimental evaluation of all the aspects that were subject to the analysis in Section 3.

Our experiments are performed on Intel Pentium IV CPU 3.6GHz processor with 1Gigabytes of DDR2 memory and a 80Gigabytes SATA hard disk, and the ORDBMS system that we used is *Oracle Release 2, version 9.2.0.1.0*.

The data set used in our experiment contains 5000 trajectories. They were generated using 1,000 real trajectories from the DOMINO project conducted by the DBMC laboratory (*www.cs.uic.edu/ wolfson/html/mobile.html*), at the UIC, which were constructed based on the Chicagoland (Cook County) electronic map. Based on these, we generated another 4000 trajectories, using road-segment endpoints from the source data-set and averaging the speed patterns the segments that existed in the real data-set, incident to each endpoint. The distribution of trajectory length in terms of number of segments per trajectory is shown in Figure 4. Assuming the average length of eight blocks to be one mile, the length of the trajectories in our data varies from 1 to 60 miles. Unless otherwise specified, the query regions were randomly generated within the area of interests to the MOD, and the time interval spans are distributed evenly throughout the lifetime of MOD. The disturbance zones that we considered were also defined by polygons in the spatial dimension and a time interval in the temporal dimension and are generated using the same method.

Recall that the metrics that we use is the *response time* for re-evaluating a set of pending continuous queries. We measure this from the time an INSERT request is specified to the TAT table (reflecting a traffic abnormality), until the answer-set (`Current_Answer` attribute in the QT table) of every pending query in the MOD is brought up-to-date.

• *Set vs. Tuple, and Before vs. After*: When is comes to the different *semantic dimensions* used in the specification of a particular trigger, we investigated two of them, each with two values: BEFORE vs. AFTER, and SET vs. TUPLE execution. Combining each of the values, we have 4 possibilities and we investigated the running times for each of them. In the experiments, we focused on one single query (consequently, one trigger) and we were varying the number of trajectories affected by the disturbance zone from 8 to

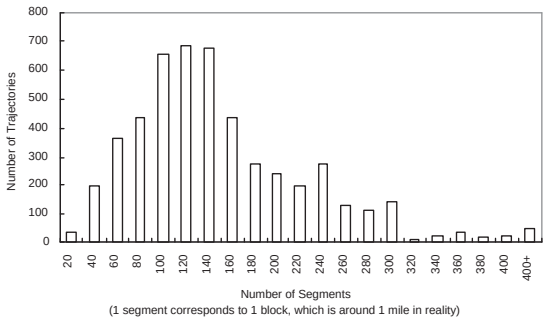

Figure 4: Trajectories Length Distribution

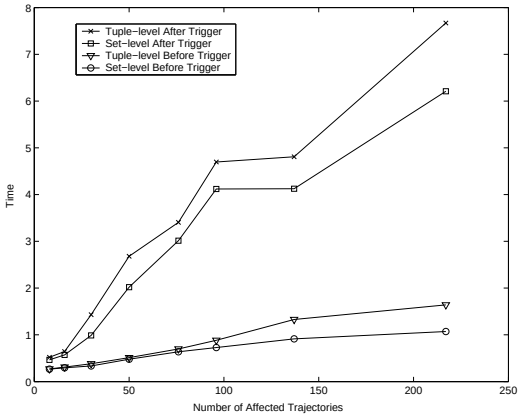

Figure 5: Semantic Dimensions of the Triggers

200, by varying the spatial location and time interval of the disturbance zone. The average response times are shown in Figure 5. As expected, the best performance is obtained when the BEFORE trigger executes in a SET oriented manner. A noteworthy observation is that loading of the *Oracle Spatial* package incurs a substantial context-switching time penalty and this is part of the reason why the discrepancy between the SET based vs. TUPLE based manner in the BEFORE context is somewhat diminished. However, the benefits of SET and BEFORE combination of semantic dimensions increase as the number of pending queries increases (c.f. Figure 9).

• *Triggers Ordering and Query Indexing*: To evaluate these approaches, we randomly selected a number of range queries that are distributed evenly in the area of interest. The number of pending queries varied from 4 to 100, and the number of affected trajectories is 100 and 200, respectively.

Figure 6 demonstrates the validity of our hypothesis that using ordering among the triggers improves the response time. However, as indicated in Figures 7 and 8, the usage of the *query index* used to intersect the MBB of the affected trajectories achieves significant improvements in the response time. This is due to the elimination of the triggers whose associated queries need not be re-evaluated. Observe that the improvements are better as the number of the pending queries increases. In Figures 7 and 8 the response time increases with the number of the trajectories affected by a given abnormality. However, the shapes (as well as the

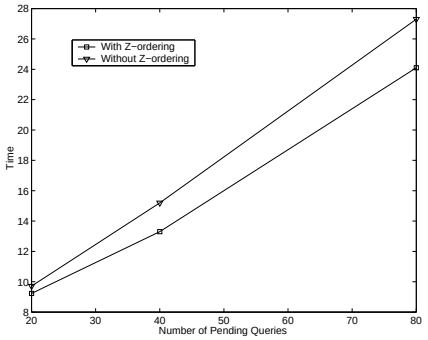

Figure 6: Z-Curve Based Ordering of Triggers

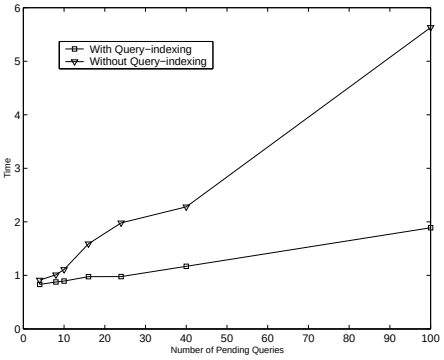

Figure 7: Query Indexing

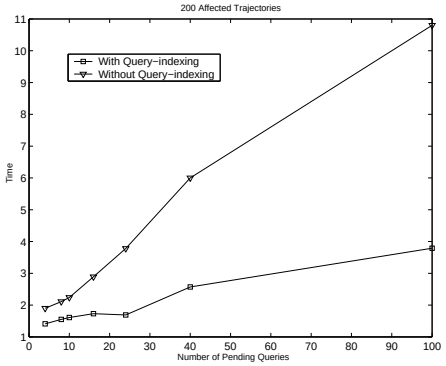

Figure 8: Query Indexing

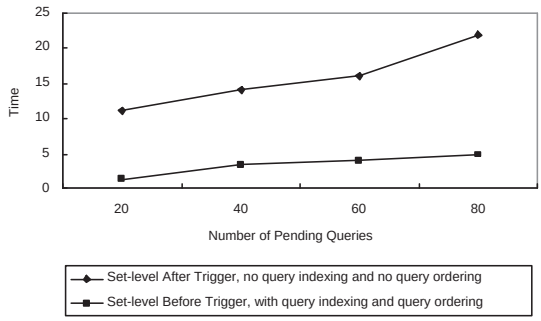

Figure 9: Potential Answers Caching

ratios) of the two curves – with and without using query indexing – follow a similar pattern.

So far, we analyzed the impact on the *response time* for each *semantic/context* dimension in isolation. In summary, Figure 9 presents our experimental observations when comparing the "naive" maintenance (i.e., TUPLE and AFTER triggers; no *query index*, no ordering among triggers) against the "full-fledge" incorporation of our methods in the system.

5. RELATED WORK AND CONCLUSIONS

The field of MOD has generated a large body of research in the past few years and a recent collection, which a large number of references is presented in [6]. There are several recent results that are similar in spirit to our work.

A body of recent work that is related to our results is affiliated with the PLACE project (*www.cs.purdue.edu/place*) The idea of indexing the queries instead of trajectories, for the purpose of efficient spatio-temporal query processing was presented in [12], and was further extended for the in-memory evaluation in [5]. Besides indexing the queries (instead of the moving objects), the works also consider the velocity constrained indexing, thus enabling the usage of incremental evaluation of queries' answers upon objects update, based on the relative locations of the objects and queries, and the assumption of maximum possible speed. Subsequently, [7, 21] utilized the concepts of locality and *No Action* regions which enables elimination of un-necessary calculations when maintaining the answers to continuous queries. The works have also addressed the problems of efficient maintenance of the NN queries and spatio-temporal joins, and have taken into consideration the settings where the regions (modelled as rectangles) are also mobile. The main source of difference between the PLACE-related works and the one presented in this paper is the very *model* of the moving objects' motion. [12, 5, 7, 21] assume that the motion is represented as a sequence of *(location,time)* updates, or a sequence of *(location, time, velocity)* updates, arriving in the MOD in a stream-like manner. As a consequence, in order to maintain the queries' answers up-to-date, some work has to be done very frequently (theoretically, upon receiving each update) and the "global" goal of is to minimize that work by avoiding un-necessary calculations. On the other hand, along the lines of the DOMINO project, we assume that the motion represented as a *trajectory* which specifies the future positions of the objects. One of the advantages of our model of motion is that queries can be posed pertaining to the *not-so-near* future, however, as a consequences a spatially-local abnormality in the parameters used in the trajectories' construction may affect the answers to many continuous queries pertaining to regions that are *not* spatially close to the region where the abnormality occurred.

A different model in which the objects' motion is represented as a collection of *dynamic* attributes of the form *(location, time, velocity)* was introduced in [14]. This model possesses some knowledge about the "near-future" whereabouts of the moving objects and the efficient processing (and maintenance) of various categories of continuous spatio-temporal queries was also investigated in some other recent works [4, 15]. Typically, a $TPR*$ tree (based on the TPR tree in [13]) is used to index the moving objects, which has the problem of "aging" of the quality of the index. The works are focused

on efficient maintenance of the queries when new moving objects are inserted or existing ones are being deleted from the MOD, however, the impact that the *bulk-updates* of the trajectories (due to unexpected updates of some values in different context dimension) has on the pending queries was not addressed. On the other hand, due to the inherent properties of the model of the motion that we adopted, dealing with bulk updates is one of the main issues for our work.

We addressed the problem of efficient maintenance of the answer-sets to continuous range queries for trajectories. Our main goal was to utilize the dependencies among the semantic/context dimensions in order to minimize computational overheads. We have implemented our system of top of an industry-strength ORDBMS (Oracle 9i) and we experimentally validated our ideas.

There are a few immediate extensions of our work. Currently, we are trying to achieve *completeness* of our system, in the manner discussed in [3, 7]. We are incorporating other categories of queries (NN and spatio-temporal joins), and we plan to consider all the possible combinations of the *dynamics* of the entities involved in the pending queries. In particular, in this work, the *reference* object (the query region) was *Static*, while the *answer objects* were *Mobile*. However, in a query like: *"Retrieve all the motels that will be within 1.5 miles from the trajectory* Tr_1*"*, the reverse holds– the *reference object* is mobile, whereas the *answer objects* are static. To say the least, for this type of queries, the *tuple/instance* oriented processing of the triggers may be more appropriate. One of our goals is to also investigate the optimization of other parameters of interest, such as average response time and system's throughput. Regardless of the model of motion adopted, the *uncertainty* is an inevitable parameter [18, 20], and we are also investigating the efficient maintenance of the pending continuous queries when the uncertainty is present in the system and a bound on the queries' error needs to be ensured. Another line of investigation, for which some preliminary results are encouraging, is to calculate a superset of the answers upon the instantaneous evaluation of the queries, and carry more computations in the main memory.

Acknowledgments: The authors would like to express their gratitude to Ouri Wolfson and his DBMC laboratory members for letting us borrow the Chicagoland trajectories data set that they designed for their DOMINO project.

6. REFERENCES

[1] Ansi/iso international standard: Database language sql. http://webstore.ansi.org.

[2] M. Breunig, C. Türker, M. Böhlen, S. Dieker, R.H. Güting, C. Jensen, L. Relly, P. Rigaux, H.-J. Schek, and M. Scholl. Architectures and implementations of spatio-temporal database management systems. In *Spatio-Temporal Databases – the Chorochronos Approach*. 2003.

[3] R. H. Güting, M. H. Bohlen, M. Erwig, C. S. Jensen, N. Lorentzos, E. Nardeli, M. Schneider, and J. R. R. Viqueira. Spatio-temporal models and languages: An approach based on data types. In *Spatio-Temporal Databases – the Chorochronos Approach*. 2003.

[4] G. Iwerks, H. Samet, and K. Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. In *Proceedings of VLDB*, 2003.

[5] D. Kalashnikov, S. Prabhakar, and S. Hambrusch. Main memory evaluation of monitoring queries over moving objects. *Distributed and Parallel Databases*, 15((2)), 2004.

[6] M. Koubarakis, T. Sellis, A.U. Frank, S. Grumbach, R.H. Güting, C.S. Jensen, N. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.-J. Scheck, M. Scholl, B. Theodoulidis, and N. Tryfona, editors. *Spatio-Temporal Databases – the CHOROCHRONOS Approach*. Springer-Verlag, 2003.

[7] M.F. Mokbel, X. Xiong, and W.G. Aref. Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *Proceedings of ACM SIGMOD*, 2004.

[8] B. Moon, H.V. Jagadish, C. Cloutsos, and J.H. Saltz. Analysis of the clustering properties of the hilbert space filling curve. *IEEE - TKDE*, 13(1), 2001.

[9] Oracle 9i. www.oracle.com/technology/products/oracle9i.

[10] N. W. Paton. *Active Rules in Database Systems*. Springer-Verlag, 1999. New York.

[11] E. Pitoura and G. Samaras. Locating objects in mobile computing. *IEEE-TKDE*, 13(4), 2001.

[12] S. Prabhakar, Y. Xia, D. Khalashnikov, W. Aref, and S. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Trans. Computers*, 51(10), 2002.

[13] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proceedings of ACM SIGMOD*, 2000.

[14] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proceedings of ICDE*, 1997.

[15] Y. Tao and D. Papadias. Spatial queries in dynamic environments. *ACM TODS*, 28(2), 2003.

[16] G. Trajcevski and P. Scheuermann. Reactive maintenance of continuous queries. *ACM SIGMOBILE Mobile Computing and Communications Review*, 8, 2004.

[17] G. Trajcevski, P. Scheuermann, O. Wolfson, and N. Nedungadi. Cat: Consistent answers to continuous queries using triggers. In *Proceedings of EDBT)*, 2004.

[18] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM TODS*, 29(3), 2004.

[19] G. Trajcevski, O. Wolfson, B. Xu, and P. Nelson. Real-time traffic updates in moving object databases. In *MDDS (in conjunction with DEXA)*, 2002.

[20] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7(3), 1999.

[21] X. Xing, M.F. Mokbel, W.G. Aref, S.E. Hambrusch, and S. Prabhakar. Scalable spatio-temporal continuous query processing for location-aware services. In *Proceedings of SSDBM*, 2004.