Efficiently Querying Moving Objects with Pre-defined Paths in a Distributed Environment *

Cyrus Shahabi, Mohammad R. Kolahdouzan, Snehal Thakkar,
Jose Luis Ambite† and Craig A. Knoblock†

Department of Computer Science and
†Information Sciences Institute
University of Southern California
Los Angeles, California 90089

[shahabi, kolahdoz, snehalth]@usc.edu [ambite, knoblock]@isi.edu

ABSTRACT

Due to the recent growth of the World Wide Web, numerous spatio-temporal applications can obtain their required information from publicly available web sources. We consider those sources maintaining moving objects with predefined paths and schedules, and investigate different plans to perform queries on the integration of these data sources efficiently. Examples of such data sources are networks of railroad paths and schedules for trains running between cities connected through these networks. A typical query on such data sources is to find all trains that pass through a given point on the network within a given time interval. We show that traditional filter+semi-join plans would not result in efficient query response times on distributed spatio-temporal sources. Hence, we propose a novel spatio-temporal filter, called *deviation filter*, that exploits both the spatial and temporal characteristics of the sources in order to improve the selectivity. We also report on our experiments in comparing the performances of the alternative query plans and conclude that the plan with spatio-temporal filter is the most viable and superior plan.

1. INTRODUCTION

The explosive growth of the Internet has made a wealth of networked information available. Much of this information is geographical, spatial, temporal, or pertains to objects that have a spatial or temporal nature. The sources of this information are heterogeneous: traditional databases with spatial extensions, geographical information systems (GIS) software packages, mapping and imagery web sites, web sites with spatial information, etc. An increasing number of web sites have information of a geospatial or temporal character. For example, detailed satellite images can be obtained from sites such as *www.terraserver.com*; maps from *www.mapquest.com*; **train schedules** from *www.amtrak.com*; geolocated points of interest such as **train stations** from *www.usgs.gov*; geographical features such as **railroad networks** from *www.nima.mil*; etc. The number of sources, the quality, and detail of the information available are continually growing all around the globe. In this paper, we focus our attention on how to efficiently query *moving objects* such as trains in a distributed environment such as the one mentioned above.

Recently there has been a growing interest in *moving object databases* that manage the spatial objects whose position changes over time [2, 3, 4, 5, 6, 7, 8]. Example applications are those who query the locations of trains, cars and planes for a given time interval. The main challenge investigated by these studies is how to model the large spatio-temporal data needed to track the position of any object at any given time (either in the past, future or now).

In this paper, we consider an environment where the content of the moving object database does not need to be modified to reflect the movement of the objects. We term this environment as "*moving objects with predefined paths and schedules*." An example application is to query the location of trains moving on a railroad network. By storing the schedules of trains' departures and arrivals, the locations of the stations and the vector data corresponding to the railroad network, we have enough information to query the location of any moving object (i.e., train) at any given time. Note that the database still needs to be modified (e.g., when schedule changes), but it does not need to be updated (and/or appended) as the objects move around the network within the provided schedule. The challenge, however, is that queries of the type of finding the location of a train in a given time interval are time consuming because of the expensive functions such as the *shortest path* function that need to be performed on large vector data as well as the

*This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC) and ITR-0082826, NASA/JPL contract nr. 961518, DARPA and USAF under agreement nr. F30602-99-1-0524, and unrestricted cash/equipment gifts from NCR, IBM, Intel and SUN.

temporal intersections that need to be applied on large sets of time intervals.

One solution to reduce the query processing time of moving objects with predefined paths and schedules is to precompute the required information and materialize it using a moving object data model such as the 3D Trajectory [3] model. This is a feasible approach if we assume that different schedules, railroads and stations information are all local and over which we have full control. However, with our assumed distributed environment, the sources of information that we would like to access are autonomous and dynamic. That is, we do not have administrative control over them, cannot modify their structure, or write data to them. The sources can change their information without warning. Different sources may contain overlapping information or only fragments of desired data.

Therefore, we propose alternative distributed query plans to realize the integration of spatial and temporal information (e.g., network of the railroads and schedules of the trains) from distributed, heterogeneous sources. We start by investigating traditional filter+semi-join plans by either applying the temporal filter first and then perform the spatial semi-join or vice versa. However, we show that there are two main drawbacks with pure filter+semi-join plans. First, spatial filters (e.g., identifying all railroad segments that overlap with a given point) are computationally complex resulting in long local or remote query processing time. Second, temporal filters (identifying all intervals that overlap with a given interval) usually have bad selectivity due to the large range of intervals covered by each instance in the temporal source (e.g., schedule table). That is, many schedules usually intersect with any query interval. Thus, temporal filters cannot effectively reduce the amount of data transferred over the network. We try to overcome the first obstacle by either performing a pre-computation step and then apply a less expensive function/filter, or delaying the spatial filter until we reduce the size of the spatial data (e.g., railroad vector data) significantly. We address the second obstacle by proposing a spatio-temporal filter (termed *deviation filter*), instead of temporal-only filters, which can also exploit the spatial characteristics of the data to improve the selectivity. Finally, we report on our experiments in evaluating and comparing the performances of our different moving object query plans. Although one version of the pre-computation approach outperforms all the other plans, the pre-computation approach may not be feasible due to limited control over remote sources. Therefore, we conclude that our deviation based approach, which is only marginally worse than the pre-computation approach, is the superior plan.

The remainder of this paper is organized as follows. We formally define the problem in Section 2. Sections 3 and 4 discuss the solutions to the problem for the centralized and distributed environments, respectively. Section 5 reports on our experimental observations. Finally, we discuss the conclusion and future work in Section 6.

2. PROBLEM DEFINITION

To better understand the problem, consider a scenario that a movie is being produced at a certain location on a certain date for several hours. The director of the movie needs to make sure that no train passes by that location while they are shooting the movie. This requires having information about the schedules of the trains running on the nearby railroads.

The following is the list of the simplified version of the sources available on the web and in our local databases, which can be used to provide necessary information for the director of the movie.

- Name of the train stations and their geographical locations (from Silicon Mapping Solutions):

  $S_{station}$ = *Stations(Station-Name, Station-Point)*

  where station-name is a character string that contains a unique name for each station and station-point represents the latitude and longitude of the station.
- Up-to-date schedule information of the trains (from the train company web site):

  $S_{schedule}$ = *Schedules(Train-ID, Departing-Station-Name, Departing-Time, Arriving-Station-Name, Arriving-Time)*
- The vector data containing the railroads' path (from Nima gazetter):

  $S_{railroads}$ = *Railroads(Railroad-ID, Railroad-Path, Starting-Point, Ending-Point)*

  where the railroad-path is a 2D line with starting-point and ending-point as its first and last vertices.

As a real-world example, the above sources of information for the United States contain around 1000 stations, 400,000 schedules, and 170,000 line segments for the railroad network. By appropriate spatio-temporal integration of the above sources, we are interested in processing the following types of queries:

- $Q1$: Find the position of a train for a given time instant.
- $Q2$: Given a geographical point and a time interval, find all the trains that pass through the point during the given time interval.

For the reminder of this paper, to simplify the discussion, we only focus on the more general query, i.e. $Q2$. We now formally define the sources of information and predicates of $Q2$ as:

- Name of the stations :
  $s = \{(s_i)|1 \leq i \leq N_s\}$ where N_s is the total number of stations.
- Locations of the stations:
  $S_{stations} = \{(s_i, (x_i, y_i))|1 \leq i \leq N_s\}$ where (x_i, y_i) specify the geographical location of the station s_i.
- Schedules of the trains:
  $S_{schedules} = \{V_k|V_k = (Train_{id}, s_i, s_j, [t_i, t_j])|1 \leq i, j \leq N_s; 1 \leq k \leq M\}$ where M is the cardinality of schedules. Note that $M \gg N_s$.
- Railroad network:
  $S_{railroads} = \{(< X_i, Y_i > ... < X_j, Y_j >)|1 \leq i, j \leq N_r; (x_i, y_i)\epsilon(X, Y)\}$ where N_r is the number of railroads in the railroad network that is in the order of N_s^2, and (X, Y) specifies the geographical location of the starting and ending points of each railroad segment. Note that not all the start and end points are stations. However, all stations are either a start or an end point or both. Hence $N_r \gg N_s$.

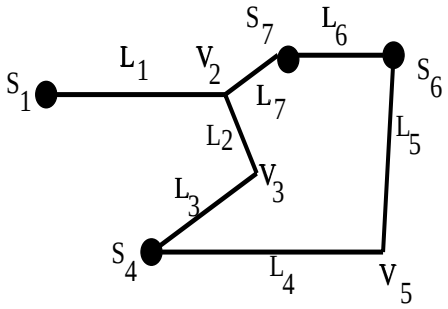


Figure 1: An example of railroad networks

- $Q_T = Intersect([t_a, t_b], [t_i, t_j])$
  Q_T, representing the temporal predicate of the query, finds schedules that have intersection with a given time interval $[t_a, t_b]$.
- $Q_S = Intersect((x, y), \Psi(S_{railroads}))$
  where $\Psi(S_{railroads})$ is a function that calculates the *shortest*[1] railroad paths between all possible **start and end point** pairs ($< X_i, Y_i >, < X_j, Y_j >$).
- $Q'_S = Intersect((x, y), \Phi(S_{railroads}, S_{stations}))$
  where $\Phi(S_{railroads}, S_{stations})$ is a function that calculates the *shortest* railroad paths between all **station** pairs ($< x_i, y_i >, < x_j, y_j >$). Since the set of the station coordinates, (x, y), is a small subset of the starting and ending points in the railroad network, (X, Y), the Φ function is computationally less expensive than Ψ.

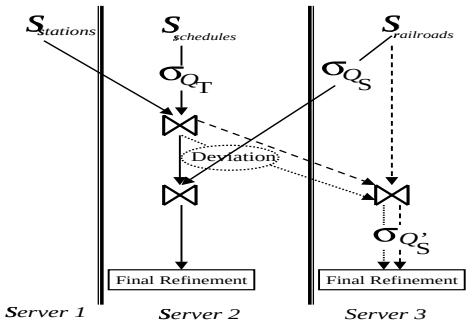
With Q_S and Q'_S, it is not sufficient to only find those line segments that intersect with (x, y), but subsequently all the paths that *include* those segments. Hence, the function Ψ constructs all paths between any possible start and end points (complexity $O(N_r^2)$) while Φ does the same operation on all possible station pairs (complexity $O(N_s^2)$). During this path construction, there may be cases where there are more than one path between two points. In these cases, both Ψ and Φ choose the shortest path, or actually never construct the other paths.

To illustrate the difference between the functions Ψ and Φ consider the following example:

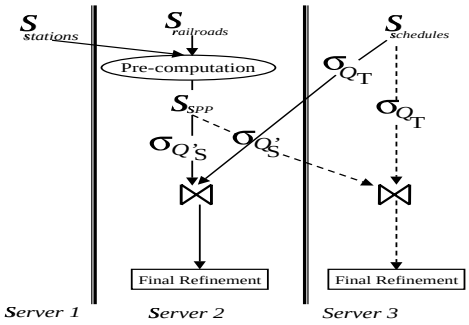
EXAMPLE 2.1.: *Figure 1 illustrates a railroad network with 7 starting and ending points, out of which 4 are stations (S_1, S_4, S_6, S_7). Table 1 shows the results of applying the Ψ and Φ functions to this network. Note that both functions only compute the shortest path between the pair of points among all possible paths. For example, there are two possible paths between S_4 and S_7. The path (L_3, L_2, L_7) is computed, while (L_4, L_5, L_6) is discarded.*

The sources $S_{stations}$ and $S_{railroads}$ contain spatial information while $S_{schedules}$ has temporal content. Q_T, Q_S and Q'_S represent the temporal and spatial predicates of the query $Q2$. Note that there are two alternative ways to evaluate spatial part of the query using either Q_S or Q'_S. Figure 2 depicts possible query plans to integrate the sources in a distributed environment.

[1]We assume that the trains travel through the shortest path between two stations as it is true in the real-world application.



a. Joining stations and schedules first.



b. Joining stations and railroads first.

Figure 2: Alternative query plans to integrate sources in a distributed environment.

3. CENTRALIZED ENVIRONMENT

In this section, we briefly describe the solution to the problem described in Section 2 using a centralized environment. We integrate the sources $S_{stations}$, $S_{schedules}$ and $S_{railroads}$ off-line using the following relational algebra expression to generate a 3D-trajectory source.

$$3D\_Trajectory(Train\_ID, Trajectory) := (S_{schedules} \bowtie_{s_i} S_{stations}) \bowtie_{<x_i, y_i>} S_{railroads} \quad (1)$$

We model each train as a moving point in 2D or equivalently a line in 3D with time as its third dimension. With this model, we can answer the queries described in Section 2 through an intersect operation on 3D-trajectory source between each train trajectory and $(x, y, [t_a, t_b])$, where (x, y) is the query point and $[t_a, t_b]$ is the given time interval. The relational algebra expression for the intersect operation is given in Equation 2. This approach has been used in the moving objects literature.

$$\sigma_{Intersect((x,y,[t_a,t_b]),Trajectory)}(3D\_Trajectory) \quad (2)$$

There are disadvantages associated with the centralized approach. The cost of building the materialized view is usually very high as the sources usually contain large sets of data. Furthermore, update and insert operations to the original sources lead to expensive updates of the 3D trajectory source. In addition, the limitations of the mediator server and/or regulations imposed by remote sources may restrict us from materializing the data locally.

4. DISTRIBUTED ENVIRONMENT

In this section, we describe four alternative approaches to integrate the sources $S_{stations}$, $S_{schedules}$ and $S_{railroads}$ in a

Result of the Ψ function	Result of the Φ function
$(L_1), (L_1, L_2), (L_1, L_2, L_3), (L_1, L_2, L_3, L_4), (L_1, L_7, L_6), (L_1, L_7)$ $(L_2), (L_2, L_3), (L_2, L_3, L_4), (L_7, L_6), (L_7)$ $(L_3), (L_3, L_4), (L_2, L_7, L_6), (L_2, L_7)$ $(L_4), (L_4, L_5), (L_3, L_2, L_7)$ $(L_5), (L_5, L_6)$ (L_6)	$(L_1, L_2, L_3), (L_1, L_7), (L_1, L_7, L_6)$ $(L_3, L_2, L_7), (L_4, L_5)$ (L_6)

Table 1: Comparison of the Ψ and Φ functions.

distributed environment. As depicted in Figure 2, the total number of query plans is more than four. However, we ignore those plans that are unrealistic due to either large data transmission over the network or expensive local/remote computations. In the following sections, we only briefly mention the unrealistic plans and the reasons we ignored them.

4.1 Temporal Filter and Spatial Semi-join ($T_F S_J$)

With this method, depicted in Figure 2a, we join the *helper source*, $S_{stations}$, with $S_{schedules}$. This is required to generate a common attribute (i.e., station coordinates) for the further join operation with $S_{railroads}$. However, the coordinates of the stations are not exploited to improve the selectivity of the temporal selection predicate.

As shown in Figure 2a, $S_{schedules}$ is first filtered and joined with the helper source to generate intermediate results. Alternative approaches can then be considered to join the intermediate results with $S_{railroads}$. One approach is to perform σ_{Q_S} on $S_{railroads}$ and transfer the results to the server that holds $S_{schedules}$. The advantage of this method is that the result of σ_{Q_S} usually constitutes a small set of data that can be quickly transmitted over the network. On the other hand, σ_{Q_S} is an expensive operation rendering this method impractical. We do not consider this method for our experiments as its efficiency is always outperformed by other methods.

The considered approach, termed $T_F S_J$, avoids expensive complexity of σ_{Q_S} by transferring the results of the temporal query to $S_{railroads}$. The advantage of this method is that we can perform a simpler spatial expression, $\sigma_{Q'_S}$, after $S_{railroads}$ and intermediate results are joined. This reduces the complexity of the query plan. However, due to the bad selectivity of the temporal predicate (specially for longer time intervals), large amount of data need to be transferred over the network that may result in longer query processing time. We consider this query plan in our experiments as a comparison point. In Section 4.2, we describe a new approach which exploits the coordinates of the helper source to improve the selectivity factor of the temporal predicate.

With this query execution plan, our query can be performed in the following steps.

- *Temporal Selection*: First, we perform a selection on the $S_{schedules}$ using the departure and arrival time and the given time interval $[t_a, t_b]$ to find all schedules that are active during the given time interval. Next, we join the selected schedules with $S_{stations}$ using station-name to include coordinates of the stations in the selected schedules.
- *Spatial Semi-join*: Next, we perform a spatial semi-join between selected schedules and $S_{railroads}$ using the coordinates of the stations and railroads.
- *Spatial Filter*: The results are then filtered using spatial selection expression $\sigma_{Q'_S}$ to select railroads that intersect with the given point (x, y).
- *Final Refinement*: Finally, we calculate the estimated time instant t at which each train reaches the given point and exclude all trains for which t has no intersection with the interval $[t_a, t_b]$.

The relational algebra expression for our query using the $T_F S_J$ query execution plan is:

$$\sigma_{FR}(\sigma_{Q'_S}(\sigma_{Q_T}(S_{schedules} \bowtie_{s_i} S_{stations}) \bowtie_{<x_i,y_i>} S_{railroads})) \quad (3)$$

4.2 Deviation Based Approach (DA)

We propose a different approach, termed *Deviation Based Approach*, which is similar to $T_F S_J$ except that it exploits the coordinates of the stations to perform a spatio-temporal selection predicate resulting in better selectivity. This approach reduces the number of the candidate schedules by filtering out the schedules that correspond to the station pairs connected through a path far from the query point (x, y).

The core idea behind this approach is as follows. First, given the coordinates of two stations, we can always compute the shortest distance (straight line) between them. However, in real-world, the railroad path between the two stations is not a straight line. Hence, we estimate how much the real path *deviates* from the straight line in the very worst case and define a term called *Path Deviation* to measure the deviation. Extending the 2D area of this deviation with the time dimension, we obtain a 3D capsule shape object. Meanwhile, the intermediate spatio-temporal tuples (station coordinates joined with schedule time intervals) can be conceptualized as straight lines in 3D space joining points (x_i, y_i, t_a) and (x_j, y_j, t_b). Consequently, the deviation filter only selects those lines that intersect with the 3D capsule shape object. This results in a much more effective filter as compared to a pure temporal filter. Meanwhile, since this filter works on the small set of point data (station coordinates) as opposed to the large set of line data (railroad vectors), it is not as computationally complex as the pure spatial filter.

DEFINITION 4.1.: Path Deviation *is defined as the ratio of the sum of the euclidian distances between the start point $P_S = (X_i, Y_i)$ and end point $P_E = (X_j, Y_j)$ with a given point (x, y), over the euclidian distance between the start and end points of a railroad segment.*

$$PD(P_S, P_E, (x, y)) = \frac{Distance(P_S,(x,y)) + (P_E,(x,y))}{Distance(P_S, P_E)} \quad (4)$$

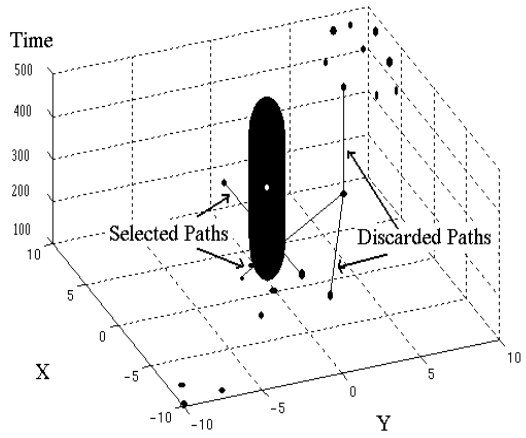


Figure 3: Graphical representation of the path deviation approach

Based on Definition 4.1, we can compute path deviation for each schedule that is active during the given time interval $[t_a, t_b]$, and exclude all schedules that have path deviation greater than a pre-defined threshold. Path deviation threshold, P_{DT}, can be pre-determined or adaptively computed during the query time. We present several algorithms to compute the path deviation threshold and prove that this approach does not result in any false drops (see Section 4.2.1).

The difference between the execution plan for this approach and the approach discussed in Section 4.1 is that the *path deviation selection predicate* is applied to the results of the temporal selection before transmission to the server which hosts $S_{railroads}$.

Path deviation selection expression ($\sigma_{Q_{DF}}$) for our query is given below.

$$\sigma_{Q_{DF}} = [PD(P_S, P_E, (x, y)) \leq P_{DT}] \quad (5)$$

The graphical representation of this approach is shown in Figure 3. Applying path deviation selection predicate is similar to creating a capsule shaped object around the given point (x, y). The length of the object depends on the given time interval $[t_a, t_b]$, while the width of the object depends on the threshold. We select the schedules for which a straight line between the starting and ending stations intersects with the capsule shaped object.

The relational algebra expression of our query using this query execution plan is:

$$\sigma_{FR}(\sigma_{Q'_S}(\sigma_{Q_{DF}}(\sigma_{Q_T}(S_{schedules} \bowtie_{s_i} S_{stations})) \bowtie_{<x_i,y_i>} S_{railroads})) \quad (6)$$

Since the path deviation predicate provides a better selectivity as compared to the temporal selection predicate, this approach results in less data transfer between the servers and better query response time.

4.2.1 Threshold Selection

In this section, we propose alternative algorithms to determine the threshold for the path deviation approach. Our first method computes a constant value for the threshold. We use Equation 7 to compute the threshold:

$$P_{DT} = max(\frac{length(railroad)}{Distance(P_S, P_E)}) \quad (7)$$

This equation computes the maximum ratio between the actual length of the railroad connecting all pairs of the starting and ending points in $S_{railroads}$ and the euclidian distance between them. We prove that this equation guarantees no false drops, i.e., no candidate trains are excluded when using this value for the path deviation threshold.

THEOREM 4.2.: *For Path Deviation Threshold P_{DT} of Equation 7, selection $\sigma_{Q_{DF}}$ results in no false drops.*

Proof: *We substitute P_{DT} in Equation 5 with the value of P_{DT} computed by Equation 7.*

$$\sigma_{Q_{DF}} = \frac{(Distance(P_S, (x, y)) + Distance(P_E, (x, y)))}{(Distance(P_S, P_E))} \leq \frac{max(length(railroad)}{Distance(P_S, P_E))} \quad (8)$$

$(Distance(P_S, (x, y)) + Distance(P_E, (x, y)))$ is the length of the shortest path, l_{SE}, between the starting and ending points that passes through the given point (x, y). Hence, if (x, y) is on a railroad path linking the starting and ending points, the length of the railroad must be greater then or equal to l_{SE}, which in turn implies that the Equation 8 is satisfied. This means that none of the railroads that intersect with (x, y) are excluded by the path deviation selection expression $\sigma_{Q_{DF}}$.

The only disadvantage of a constant value for the threshold is that a large value of the path deviation for some railroads results in a large value of the overall threshold for the entire network. For example, in real-world this scenario happens if a railroad track is not a straight line between two stations (shortest distance) due to some natural limitations (e.g., existence of a lake in the way). The larger the value of the threshold, the worse the selectivity of the path deviation selection predicate. This means that guaranteing no false drops in path deviation approach (i.e., satisfying Equation 8) may result in selecting and transferring more schedules.

4.2.2 Adaptive Threshold

We propose two approaches to adaptively select the value of the threshold.

1. With the first approach, we divide the spatial data (e.g., $S_{railroads}$) into distinct geographical regions and calculate different threshold values for each region using Equation 6. In real-world, for a flat area such as a desert, this would result in less conservative thresholds while for an area with several mountains, a more conservative threshold will be chosen. Subsequently, different path deviation predicates with different thresholds are applied to different regions. This results in better selectivity for the path deviation selection predicate as the high value of the threshold for a particular region does not affect the regions with lower values of the threshold.

2. With the second approach, we consider railroad paths with higher values of threshold as exceptions and exclude them from the computation of the overall threshold. The excluded paths will then be considered as possible candidates and are examined through the final refinement.

If the conservative value for the threshold is due to existence of different regions with different railroad characteristics, then the first approach should be chosen. However, if the large value of the computed threshold for a given network is due to a small number of anomalies, then the second approach is more promising. Trivially, one can consider a hybrid approach to capture both anomalies and regional trends, simultaneously.

4.3 Pre-computation & Temporal Filter (PT_F)

With this approach, we propose to exploit the coordinates of the stations (as subset of the start and end points in $S_{railroads}$), to pre-compute all possible railroad paths between all the station pairs and generate a pre-computed source, termed S_{spp}. The size of S_{spp} is $O(N_s^2)$ (N_s is number of the stations), which is usually greater than the number of the railroads in the network. However, the simpler spatial expression $\sigma_{Q'_S}$ is now applicable to the S_{spp} that reduces the complexity of spatial filter.

The *pre-computation and temporal filter* approach utilizes S_{spp} to perform the spatial expression $\sigma_{Q'_S}$. Unlike the approach discussed in Section 4.1, the spatial predicate is performed before the join operation between S_{spp} and the result of the selection on $S_{schedules}$. This has the advantage that the spatial expression $\sigma_{Q'_S}$ can be applied earlier to S_{spp}. However, this approach suffers from the bad selectivity of the temporal selection predicate applied to $S_{schedules}$ which results in large data transfer over the network.

The relational algebra expression to perform our query with this approach is:

$$\sigma_{FR}(\sigma_{Q_T}(S_{schedules}) \bowtie_{s_i} (\sigma_{Q'_S} S_{spp})) \quad (9)$$

4.4 Pre-computation & Spatial Filter (PS_F)

With this approach, we eliminate the problem of bad selectivity of the temporal predicate of (PS_T). We transmit the results of the spatial selection expression $\sigma_{Q'_S}$ applied to S_{spp} to the server that holds $S_{schedules}$.

With this method, we first perform a spatial selection on S_{spp} to find all railroad paths that intersect with the given point (x, y). At the same time, we perform a temporal selection on $S_{schedules}$ to find all schedules that are active during the given time interval $[t_a, t_b]$. Next, we transfer the results of the spatial selection expression $\sigma_{Q'_S}$ to the server that contains $S_{schedules}$. The join between the selected railroad paths and schedules finds all candidate results of the query. Finally, we perform the final refinement step as described in the previous approaches.

The relational algebra expression to perform our query with this approach is:

$$\sigma_{FR}((\sigma_{Q'_S} S_{spp}) \bowtie_{s_i} \sigma_{Q_T}(S_{schedules})) \quad (10)$$

With this approach, the complexity of the spatial predicate is reduced as we use $\sigma_{Q'_S}$ on S_{ssp}. In addition, the effect of the bad selectivity of the temporal predicate is eliminated by transferring the selected railroads over the network as opposed to the selected schedules. Our experimental results provided in Section 5.2 show that this approach outperforms all the previous approaches. However, this approach is based on the assumption that full access control over the servers is granted. In a distributed environment (e.g., WWW) with limited access control (i.e., read-only access), creating a new source (i.e., S_{spp}) in a remote server may not be possible which renders this approach impractical.

5. PERFORMANCE EVALUATION

In this section, we describe the experimental setup and results for our different query execution plans.

5.1 Implementation

The configuration for the experiments consisted of two SUN servers connected through a 100 mpbs LAN. Both servers were Ultra Enterprise 250 with two Sparc II processors and 512MB RAM and were running Solaris 2.6. Both systems run Informix Universal Server 9.2 with ESRI Spatial datablade.

We utilized a set of synthetically generated data for our experiments. The data was generated to be as similar to the real-world data as possible. The ratio of the stations, schedules and railroads correspond to the real-world example described in Section 2.

The location of the stations were uniformly selected across the globe. Random railroad segments connect each station with its 3 to 16 closest stations to provide different connectivity for stations. In addition, between 10 to 30 random schedules were generated for pairs of stations that were connected through a railroad segment.

We used Java and JDBC to manipulate data with Informix database servers on both servers. The Java program always resided on the server that performed the final refinement operation. We varied the given time interval $[t_a, t_b]$ from 15 minutes to 120 minutes with 15 minute increments. Different values of start time for the given time interval, t_a, were randomly selected. We run the experiments several times to obtain the average query response time for different time intervals. Running the experiments for different number of stations (i.e., 100, 500 and 1000) showed the same trend. Section 5.2 presents the results for 500 stations.

5.2 Experimental Results

Figure 4 depicts the results of our experiments for the four alternative query plans discussed in Section 4. In this figure, the X-axis represents the range of the given time interval, varying from 15 to 120 minutes. The Y-axis shows the average query response time in seconds.

As illustrated, one of the pre-computation based approaches, PS_F, outperforms all the other approaches. This is because with this plan, the complex σ_{Q_S} is replaced with the less expensive $\sigma_{Q'_S}$ (which works on the pre-computed S_{spp}). Besides being less complex, $\sigma_{Q'_S}$ has also a very good selectiv-

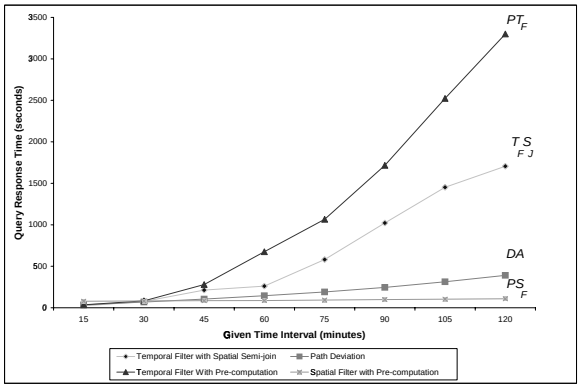


Figure 4: Comparison between the four query plans

ity, which translates to low network data transfer. Moreover, the temporal filter σ_{Q_T} is applied at the local server avoiding the transmission of bulky schedule data over the network. However, this approach assumes complete control over the remote server hosting $S_{schedules}$, because both S_{spp} and the results of the selection predicate over S_{spp} must be stored at the remote server. This assumption may not be a valid one when the control over remote servers is restricted.

The next best plan is our deviation based approach DA for query time intervals longer than 15 minutes. The reason is that for long ranges of time intervals the path deviation filter has better selectivity than the temporal filter. Moreover, since the spatial filter can be delayed until $S_{railroads}$ is joined with the other two sources, a less complex $\sigma_{Q'_S}$ substitutes the complex σ_{Q_S}.

Finally, the pure temporal-filter+spatial-join plan, $T_F S_J$, only performs well for small time intervals. This is because of the good selectivity of this filter only for short ranges of time interval. Similarly, PT_F does not perform well for the exact same reason. Note that although both PT_F and $T_F S_J$ do not utilize the expensive σ_{Q_S} (because of PT_F's pre-computation step and $T_F S_J$'s delayed spatial operation), the overhead of the temporal data transfer over the network is so high that cancels out all the low complexity benefits of $\sigma_{Q'_S}$.

6. CONCLUSION AND FUTURE WORK

We focused on an application that queries moving objects with predefined paths and schedules. Although the modeling aspect of this type of moving object databases may not be as challenging as the unrestricted ones, its query processing becomes challenging in a distributed environment. We demonstrated that the complexity of the spatial selection operation and bad selectivity of the temporal selection operation render pure filter+semi-join query plans inefficient. We investigated solutions to both problems by first joining each source with a helper spatial source and then 1) performing pre-computation on the combination of two spatial sources, or 2) replacing temporal filters with spatio-temporal ones on the combination of the spatial and the temporal sources. The pre-computation approach may not be feasible in some distributed environments (e.g., WWW) where there is only limited control (e.g., read-only access) on the remote sources. Therefore, our deviation based approach, which performs the filtering using both spatial and temporal characteristics simultaneously, becomes the most viable and superior approach. Our experimental results demonstrated that the query response time of the deviation approach is on average $\frac{1}{3}$ of that of the temporal-filter+spatial-join approach.

We plan to extend this study in three ways. First, we would like to perform more experiments with the real-world data sets. It would be interesting to see how much moving object queries on real data would benefit from adaptive threshold computation of the deviation approach as compared to a single constant threshold. We also intend to investigate querying of moving objects with unrestricted paths and schedules, such as cars with GPS devices, in our distributed environment. Finally, we want to incorporate our deviation based query plan into the WorldInfo Assistant [1] application and show its usefulness for efficient query evaluation.

7. REFERENCES

[1] J. L. Ambite, C. A. Knoblock, M. R. Kolahdouzan, M. Muslea, C. Shahabi, and S. Thakkar. The WorldInfo Assistant: An application for spatio-temporal information integration on the web. *27th International Conference on Very Large Data Bases (Demonstration), September 11-14, 2001, Roma, Italy.*

[2] M. Erwig, R. H. Güting, M. Schneider, and M. Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *GeoInformatica*, 3:269–296, 1999.

[3] L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, 29(2):319–330, 2000.

[4] D. Peuquet and N. Duan. An event-based spatiotemporal data model (ESTDM) for temporal analysis of geographical data. *International Journal of Geographical Information Systems*, pages 7–24, 1995.

[5] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 422–432, 1997.

[6] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Querying the uncertain position of moving objects. *Temporal Databases: Research and Practice. (the book grow out of a Dagstuhl Seminar, June 23-27, 1997)*, 1399:310–337, 1998.

[7] N. Tryfona and T. Hadzilacos. Logical data modelling of spatio temporal applications: Definitions and a model. *IDEAS, International Database Engineering and Applications Symposium*, pages 14–23, 1998.

[8] O. Wolfson, A. P. Sistla, B. Xu, J. Zhou, and S. Chamberlain. Domino: Databases for moving objects tracking. *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*, pages 547–549, 1999.