Indexing Mobile Objects Using Duality Transforms

Dimitris Papadopoulos
UC Riverside
dimitris@cs.ucr.edu

George Kollios
Boston University
gkollios@cs.bu.edu

Dimitrios Gunopulos
UC Riverside
dg@cs.ucr.edu

Vassilis J. Tsotras
UC Riverside
tsotras@cs.ucr.edu

Abstract

We present techniques to index mobile objects in order to efficiently answer range queries about their future positions. This problem appears in real-life applications, such as predicting future congestion areas in a highway system, or allocating more bandwidth for cells where high concentration of mobile phones is impending. We address the problem in external memory and present dynamic solutions, both for the one-dimensional, as well as the two-dimensional cases. Our approach transforms the problem into a dual space that is easier to index. Finally we discuss advantages and disadvantages among the various schemes proposed in literature for indexing mobile objects.

1 Introduction

Applications providing location-based services, such as traffic monitoring, intelligent navigation, and mobile communications management, fail to be adequately supported by traditional database management systems. The assumption that data stored in the database remain constant, unless explicitly updated, is the foundation of a model where updates are issued in discrete steps. On the other hand, the above applications deal with continuously changing attributes [21, 26], as for example the object locations. If a DBMS were to update such dynamic attributes every unit of time, it would entail a prohibitively high update overhead.

An elegant solution to address the problem is to use a function of time $f(t)$ to abstract the location of a mobile object. The current location of a moving object at any time instant can then be easily calculated, since an update has to be issued only when the parameters of f change (e.g. speed, direction) [21, 25, 14, 17, 1, 3]. Clearly, this approach reduces the update overhead. Nevertheless, it presents novel challenges, such as the need for appropriate data models, query languages, and query processing and optimization techniques.

This paper considers the problem of indexing mobile objects. We are interested in answering range queries over the objects' future locations. In particular, we present efficient indexing techniques based on the duality transformation. Both the one-dimensional (moving on a line) and two-dimensional (moving on the plane) cases are discussed.

Section 2 provides a formal problem description, while Section 3 describes the duality transformation. The 1-dimensional case is addressed in Section 4. Section 5 introduces the 1.5-dimensional problem, which is a restricted, yet very interesting, version of the 2-dimensional case. The technique for indexing objects that move freely in two dimensions is illustrated in Section 6. Related work and discussion pointing out advantages and disadvantages of the methods that employ indexing techniques in the primal space and the dual space follows in Section 7. Finally, Section 8 concludes the paper.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

2 Problem description

Consider a database that records the positions of moving objects in two dimensions on a finite terrain. For simplicity, we assume that objects move with a constant velocity vector starting from a specific location at a specific time instant. This is enough for calculating the future position of the object, provided that the characteristics of its motion remain the same. Objects update their motion information when their speed and direction change. Moreover, the system is dynamic, i.e. objects may be deleted or new objects may be inserted.

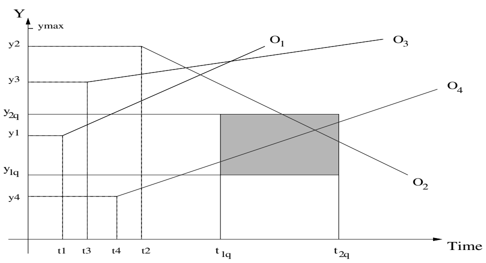

Figure 1: Trajectories and query in (t, y) plane.

Let $P(t_0) = [x_0, y_0]$ be the initial position at time t_0. Then, its position at time $t > t_0$ is $P(t) = [x(t), y(t)] = [x_0 + v_x(t - t_0), y_0 + v_y(t - t_0)]$, where $V = [v_x, v_y]$ is its velocity vector. Figure 1 depicts the projection of the object trajectories on the (t, y) plane.

We would like to answer queries of the form: "Report the objects located inside the rectangle $[x_{1q}, x_{2q}] \times [y_{1q}, y_{2q}]$ at the time instants between t_{1q} and t_{2q} (where $t_{now} \leq t_{1q} \leq t_{2q}$), given the current motion information of all objects" (This is called the *two dimensional MOR query* in [14]).

We make the assumption that moving objects have velocities in $[v_{min}, v_{max}]$. This is a realistic assumption; for example objects moving slower than v_{min} can be considered (for all practical purposes) "static" and examined separately. Similarly, in a typical scenario objects have some maximum speed.

3 The dual space-time representation

Figure 1 corresponds to the straightforward approach of representing a moving object, i.e. by plotting its trajectory as a line in the time-location (t, y) plane (same for (t, x) plane). The equation describing each line is $y(t) = vt + a$ where v is the slope (velocity in this case) and a is the intercept, which is computed using the motion information. In this setting, also termed as the "primal" space, the query is expressed as the 2-dimensional interval $[(y_{1q}, y_{2q}), (t_{1q}, t_{2q})]$, and it reports the objects that correspond to the lines intersecting the query rectangle.

The general duality transform maps a hyper-plane h from R^d to a point in R^d and vice-versa. For example, a line with equation $y(t) = vt + a$ from the primal plane (t, y) is mapped to a point (v, a) in the dual plane, where one axis represents the velocity v and the other the intercept a (this is called Hough-X transform in [12]). Accordingly, the 1-d query $[(y_{1q}, y_{2q}), (t_{1q}, t_{2q})]$ becomes a polygon in the dual space. By using a linear

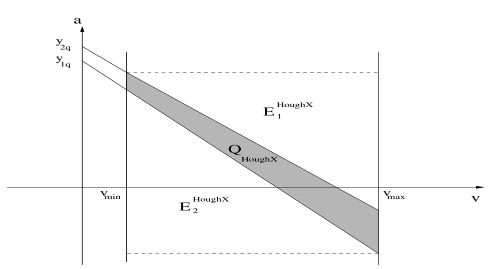

Figure 2: Query on the dual Hough-X plane.

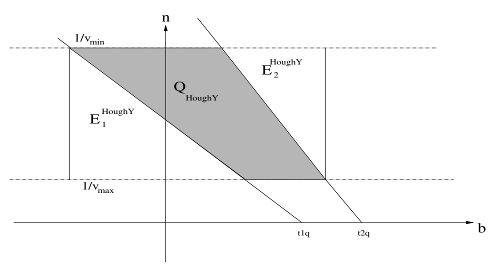

Figure 3: Query on the dual Hough-Y plane.

constraint query [8], the query Q in the dual Hough-X plane (Figure 2) is expressed in the following way [14]:

If $v > 0$, then $Q = C_1 \wedge C_2 \wedge C_3 \wedge C_4$, where:
$C_1 = v \geq v_{min}$, $C_2 = v \leq v_{max}$,
$C_3 = a + t_{2q}v \geq y_{1q}$ and $C_4 = a + t_{1q}v \leq y_{2q}$

If $v < 0$, then $Q = D_1 \wedge D_2 \wedge D_3 \wedge D_4$, where:
$D_1 = v \leq -v_{min}$, $D_2 = v \geq -v_{max}$,
$D_3 = a + t_{1q}v \geq y_{1q}$ and $D_4 = a + t_{2q}v \leq y_{2q}$

A class of transforms with similar properties may be used for the mapping. The problem setting parameters determine which one is more useful. For example, by rewriting the equation $y = vt + a$ as $t = \frac{1}{v}y - \frac{a}{v}$, we can arrive to a different dual representation. The corresponding point in this dual plane has coordinates (n, b), where $n = \frac{1}{v}$ and $b = -\frac{a}{v}$ (called the Hough-Y transform in [12]). Coordinate b represents the point where the line intersects the line $y = 0$ in the primal space. Note that the Hough-Y transform cannot represent horizontal lines. Similarly, the Hough-X transform cannot represent vertical lines. Nevertheless, since in our setting lines have a minimum and maximum slope (velocity is bounded by $[v_{min}, v_{max}]$), both transforms can be used.

4 Indexing in one dimension

By using the dual space-time representation, the problem of indexing mobile objects on a line is transformed into the problem of *simplex* range searching in two dimensions. In simplex range searching we are given a set S of 2-dimensional points, and we want to answer efficiently queries of the following form: given a set of linear constraints $ax \leq b$, find all points in S that satisfy all the constraints. Geometrically, the constraints form a polygon on the plane, and we want to find the points in the interior of the polygon. In [14] it was shown that simplex reporting in d-dimensions with a query time of $O(n^{\delta} + \kappa)$ I/O's, where N is the number of points, $n = N/B$, K is the number of reported points, $k = K/B$, and $0 < \delta \leq 1$, requires $\Omega(n^{d(1-\delta)-\epsilon})$ disk blocks, for any fixed ϵ.

A corollary of this lower bound is that in the worst case a data structure that uses linear space to answer the 2-dimensional simplex range query and thus the 1-dimensional MOR query, requires $O(\sqrt{n} + k)$ I/O's. In [14] an almost optimal solution based on partition trees [15] was presented. That solution achieves $O(n^{\frac{1}{2}+\epsilon} + k)$ query time using linear space, but the hidden constant factor becomes large even for small ϵ.

As a more practical approach (with good average query performance), [14] proposed to index the dual points using a point access method (PAM). Even though PAMs were designed to answer *orthogonal* range queries, Goldstein et al. [8] have proposed an algorithm to answer simplex range search using R-trees. This is accomplished by changing the search procedure of the tree. Apart from the R-tree family, this method can be applied to other access methods. In particular, [14] suggests using a k-d-tree like structure to index the 1-dimensional problem in the Hough-X (or Hough-Y) space, for an expected logarithmic query time.

A different approach is based on a *query approximation* idea using the Hough-Y dual plane. In general, the b coordinate can be computed at different horizontal $(y = y_r)$ lines. The query region is described by the intersection of two half-plane queries (Figure 3). The first line intersects the line $n = \frac{1}{v_{max}}$ at the point $(t_{1q} - \frac{y_{2q}-y_r}{v_{max}}, \frac{1}{v_{max}})$ and the line $n = \frac{1}{v_{min}}$ at the point $(t_{1q} - \frac{y_{2q}-y_r}{v_{min}}, \frac{1}{v_{min}})$. Similarly the other line that defines the query intersects the horizontal lines at $(t_{2q} - \frac{y_{1q}-y_r}{v_{max}}, \frac{1}{v_{max}})$ and $(t_{2q} - \frac{y_{1q}-y_r}{v_{min}}, \frac{1}{v_{min}})$, respectively.

Since access methods are more efficient for rectangular queries, suppose that we approximate the simplex query with a rectangular one. In Figure 3 the query rectangle will be $[(t_{1q} - \frac{y_{2q}-y_r}{v_{min}}, t_{2q} - \frac{y_{1q}-y_r}{v_{max}}), (\frac{1}{v_{max}}, \frac{1}{v_{min}})]$. Note that the query area is enlarged by the area $E^{HoughY} = E_1^{HoughY} + E_2^{HoughY}$ which is computed as:

$$E^{HoughY} = \frac{1}{2}(\frac{v_{max} - v_{min}}{v_{min} \times v_{max}})^2(\mid y_{2q} - y_r \mid + \mid y_{1q} - y_r \mid) \quad (1)$$

The objective is to minimize E^{HoughY} since it represents a measure of the extra I/O's that an access method will have to perform for solving an 1-dimensional MOR query. E^{HoughY} is based on both y_r (i.e. where the b coordinate is computed) and the query interval (y_{1a}, y_{2q}) which is unknown. Hence, c indices are kept (where c is a small constant) at equidistant y_r's. All c indices contain the same information about the objects, but use different y_r's. The i-th index stores the b coordinates of the data points using $y = \frac{y_{max}}{c} \times i, i = 0, \ldots, c-1$. Conceptually, y_i serves as an "observation" element, and its corresponding index stores the data as observed from position y_i. A given 1-dimensional MOR query will be forwarded to the index(es) that minimize E^{HoughY}. Since all 2-dimensional approximate queries have the same rectangle side $(\frac{1}{v_{max}}, \frac{1}{v_{min}})$, the rectangle range search is equivalent to a simple range search on the b coordinate axis. Thus each of the c "observation" indices can simply be a B+-tree [7]. [14] shows that the query can be answered with bounded error in logarithmic time.

5 The 1.5-dimensional problem

When the mobile objects are restricted to move on a given collection of routes (roads) on the finite terrain, an interesting case of the general 2-dimensional problem is formed, namely the 1.5-dimensional problem. There is a strong motivation for such an environment: for the applications we consider, objects (cars, airplanes) move through a network of predefined routes (freeways, airways).

The 1.5-dimensional problem can naturally be reduced to a collection of 1-dimensional queries. Specifically, each predefined route can be represented as a sequence of connected line segments. The positions of these line segments are indexed by a standard SAM. The extra cost for maintaining this index is negligible, since: (i) there are far less routes than moving objects, (ii) each route can be approximated by a small number of line segments, and, (iii) new routes are not introduced frequently. We apply the techniques for the 1-dimensional case in order to index the objects moving on a segment of a given route.

In order to answer the two dimensional MOR query, the SAM described above identifies the intersection of the routes with the query's spatial predicate, i.e. the rectangle $[x_{1q}, x_{2q}] \times [y_{1q}, y_{2q}]$. Since each route is modeled as a sequence of line segments, the intersection is also a set of line segments, possibly disconnected. Each such intersection corresponds to the spatial predicate of an 1-dimensional query for this route. In this setting we assume that when routes intersect, objects remain in the route previously traveled (otherwise an update is issued).

6 Indexing in two dimensions

The general 2-dimensional problem (Figure 4) is addressed by decomposing the motion of the object into two independent motions, one in the (t, x) plane and one in the (t, y) plane. Each motion is indexed separately. Next we present the procedure used in order to build the index, as well as the algorithm for answering the 2-d query.

6.1 Building the index

We begin by decomposing the motion in (x, y, t) space into two motions on the (t, x) and (t, y) plane.

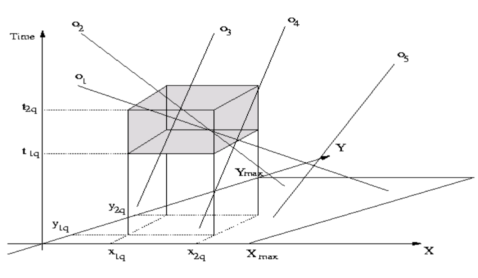

Figure 4: Trajectories and query in (x, y, t) space.

Furthermore, on each projection, we partition the objects according to their velocity. Objects with small velocity magnitude are stored using the Hough-X dual transform, while the rest of them are stored using the Hough-Y transform, i.e into distinct index structures.

The reason for using different transforms is that motions with small velocities in the Hough-Y approach are mapped into dual points (n, b) having large n coordinates ($n = \frac{1}{v}$). Thus, since few objects have small velocities, by storing the Hough-Y dual points in an index structure such an R*-tree, MBR's with large extents are introduced, and the index performance is severely affected. On the other hand, by using a Hough-X index for the small velocities' partition, we eliminate this effect, since the Hough-X dual transform maps an object's motion to the (v, a) dual point.

When a dual point is stored in the index responsible for the object's motion in one of the planes, i.e. (t, x) or (t, y), information about the motion in the other plane is also included. Thus, the leaves of both indices for the Hough-Y partition store the record (n_x, b_x, n_y, b_y). Similarly, for the Hough-X partition, in both projections, we keep the record (v_x, a_x, v_y, a_y). In this way, the query can be answered using one of the indices; either the one responsible for the (t, x) or the (t, y) projection.

On a given projection, the dual points (i.e. (n, b) and (v, a)) are indexed using R*-trees [4]. The R*-tree has been modified in order to store points at the leaf level, and not degenerated rectangles. Therefore, we can afford storing extra information about the other projection. An outline of the procedure for building the index follows:

1. Decompose the 2-d motion into two 1-d motions on the (t, x) and (t, y) planes.
2. For each projection, build the corresponding index structure. To do so, partition the objects according to their velocity: Objects with small velocity are stored using the Hough-X dual transform, while the rest are stored using the Hough-Y dual transform. Motion information about the other projection is also included.

In order to pick one of the two projections and answer the simplex query, we use the techniques described next.

6.2 Answering the query

The 2-dimensional MOR query is mapped to a simplex query in the dual space. The query is the intersection of four 3-d hyperplanes, while the projection of the query to (t, x) and (t, y) planes are wedges, as in the 1-dimensional case.

A given a 2-dimensional MOR query, is first decomposed into two 1-dimensional queries, one for each projection. Furthermore, on a given projection, the simplex query is asked in both partitions, i.e. Hough-Y (for fast objects) and Hough-X (for slow objects).

On the Hough-Y plane the query region is given by the intersection of two half-plane queries, as shown in Figure 3. Consider the parallel lines $n = \frac{1}{v_{min}}$ and $n = \frac{1}{v_{max}}$. As illustrated in section 4, if the simplex query was answered approximately, the query area would be enlarged by $E^{HoughY} = E_1^{HoughY} + E_2^{HoughY}$ (the triangular areas in Figure 3). Also, let the actual area of the simplex query be Q^{HoughY}. Similarly, on the dual Hough-X plane (Figure 2), let Q^{HoughX} be the actual area of the query, and E^{HoughX} be the enlargement. The algorithm chooses the projection which minimizes the following criterion κ:

$$\kappa = E^{HoughY}/Q^{HoughY} + E^{HoughX}/Q^{HoughX} \tag{2}$$

Since the whole motion information is kept in the indices, it is used in order to produce the exact result set of objects. An outline of the algorithm for answering the exact 2-dimensional MOR query follows:

1. Decompose the query into two 1-d queries, for the (t, x) and (t, y) projection.
2. Get the dual query for each projection (i.e. the simplex query).
3. Calculate the criterion κ for each projection, and choose the one (say π) that minimizes it.
4. Answer the query by searching the Hough-X and Hough-Y partition, using projection π.
5. Put an object in the result set, only if it satisfies the query.
6. Use the full motion information to do the filtering "on the fly".

7 Related work and discussion

While intuitive, the space-time representation (on the "primal" space) is problematic, since trajectories correspond to long lines. Long lines are difficult to index efficiently with traditional indexing techniques. Consider for example using a Spatial Access Method, such an R-tree [11] or an R*-tree [4]. Here each line is approximated by a minimum bounding rectangle (MBR). Obviously, the MBR approximation has much larger area than the line itself. Furthermore, since the trajectory of an object is valid until an update is issued, it has a starting point but no end. Thus all trajectories expand till "infinity", i.e. they share an ending point on the time dimension.

Another approach is to partition the space into disjoint cells and store in each cell those lines that intersect it [24, 6]. This could be accomplished by using an index such an R+-tree [20], a cell-tree [10], and the PMR-quadtree [19]. The shortcoming of this solution is that it introduces data replication, since each trajectory is copied into all cells that intersect it. Moreover, using space partitioning would also result in high update overhead, since when an object changes its motion information, it has to be removed from all cells that stores its trajectory.

An interesting alternative for efficient indexing of mobile objects in the primal space was proposed by Saltenis et al. [17]. They introduced the time-parameterized R-tree (TPR-tree), which extends the R*-tree. The coordinates of the bounding rectangles in the TPR-tree are functions of time and, intuitively, are capable of following the objects as they move. The position of a mobile object is represented by its location at a particular time instant (reference position) and its velocity vector.

In [3] a main memory framework (kinetic data structure) was proposed and addresses the issue of mobility and maintenance of configuration functions among continuously moving objects. Application of this framework to external range trees [2] appears in [1].

Other related work considers nearest neighbor queries in a mobile environment [13, 9, 22], time-parameterized queries [23] and selectivity estimation for moving object range queries [5].

An important characteristic of the TPR-tree and the duality transforms is that they have linear space requirements. We now illustrate the advantages and disadvantages among these two approaches. Experimental results, for the two dimensional case, between them are presented in [16] and are not included here for brevity.

The performance of the TPR-tree highly depends on the knowledge of the characteristics of the workload. The average time interval between updates (UI) and the average length of the temporal part of the queries (W) have to be given as parameters to the index beforehand. These two parameters define the horizon $H = UI + W$. The TPR-tree is optimized for answering queries whose temporal part lies within this horizon (it actually provides the best query performance for queries within the horizon [16]). It also recalculates and updates its time-parameterized MBRs whenever an update is issued. This is necessary for the TPR-tree, since the size of the bounding regions increases over time. If these parameters are not known in advance (for example in applications where UI or W cannot be easily predicted), [18] has proposed to use the automatic horizon estimation feature. Nevertheless, the query performance of the TPR-tree worsens.

On the other hand, the duality transform method has larger space requirements, but better update performance. In our experiments [16], the duality transform uses about double the space, but has about 45% faster updates. The query performance was comparable (on average 20% more) for queries inside the horizon, but was much faster than the TPR-tree performance for queries outside the horizon. However, the duality transforms cannot easily accommodate 3-d datasets, as the TPR-tree does. This is because, in the three-dimensional setting, the movement of an object would have to be decomposed into three independent movements, one for each projection on the planes (t, x), (t, y) and (t, z), in order for the dual transformation method to be applied.

8 Conclusions

We examined the problem of indexing mobile objects using dual transformations. We considered both the 1-dimensional and 2-dimensional cases, and presented external memory indexing techniques, in order to efficiently answer range queries about the objects' future locations. An interesting future direction of research is joins among relations of mobile objects. Furthermore, it would be interesting to study how the mobile objects' agility (i.e. how often updates are issued) affects the performance and robustness of the indexing methods.

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing Moving Points. *In Proc. PODS*, pp. 175–186, 2000.

[2] L. Arge, V. Samoladas, and J.S. Vitter. On Two-Dimensinal Indexability and Optimal Range Search Indexing. In *Proc. PODS*, pp. 346–357, 1999.

[3] J. Basch, L. Guibas, and J. Hershberger. Data Structures for Mobile Data. In *Proc. of the 8th ACM-SIAM Symposium on Discrete Algorithms*, pp. 747–756, 1997.

[4] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proc. ACM SIGMOD*, pp. 322–331, 1998.

[5] Y.-J. Choi and C.-W. Chung. Selectivity Estimation for Spatio-Temporal Queries to Moving Objects. In *Proc. ACM SIGMOD*, 2002 (to appear).

[6] H. D. Chon, D. Agrawal, and A. El Abbadi. Query Processing for Moving Objects with Space-Time Grid Storage Model. In *Proc. MDM*, pp. 121–130, 2002.

[7] D. Comer. The Ubiquitous B-Tree. *Computing Surveys*, 11(2):121–137, June 1979.

[8] J. Goldstein, R. Ramakrishnan, U. Shaft, and J.B. Yu. Processing Queries By Linear Constraints. In *Proc. PODS*, pp. 257–267, 1997.

[9] D. Gunopulos, G. Kollios, and V. J. Tsotras. All-Pairs Nearest Neighbors in a Mobile Environment. In *7th Hellenic Conference on Informatics*, Ioannina, Greece, August 1999.

[10] O. Gunther. The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases. In *Proc. ICDE*, 1989.

[11] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proc. ACM SIGMOD*, pp. 47–57, 1984.

[12] H. V. Jagadish. On Indexing Line Segments. In *Proc. VLDB*, pp. 614–625, 1990.

[13] G. Kollios, D. Gunopulos, and V. Tsotras. Nearest Neighbor Queries in a Mobile Environment. In *Proc. of the Spatio-Temporal Database Management Workshop, Edinburgh, Scotland*, pp. 119–134, 1999.

[14] G. Kollios, D. Gunopulos, and V. Tsotras. On Indexing Mobile Objects. In *Proc. PODS*, pp. 261–272, 1999.

[15] J. Matousek. Efficient Partition Trees. *Discrete and Computational Geometry*, 8:432–448, 1992.

[16] D. Papadopoulos, G. Kollios, D. Gunopulos, and V.J. Tsotras. Indexing Mobile Objects on the Plane. In *Proc. 5th International Workshop on Mobility in Databases and Distributed Systems*, 2002, (to appear).

[17] S. Saltenis, C. S. Jensen, S. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proc. ACM SIGMOD*, pp. 331–342, May 2000.

[18] S. Saltenis and C. S. Jensen. Indexing of Moving Objects for Location-Based Services. In *Proc. ICDE*, pp. 463–472, 2002.

[19] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison Wesley, June 1990.

[20] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proc. VLDB*, pp. 507–518, 1987.

[21] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proc. ICDE*, pp. 422–432, 1997.

[22] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *Proc. SSTD*, pp. 79–96, 2001.

[23] Y. Tao and D. Papadias. Time-Parameterized Queries in Spatio-Temporal Databases. In *Proc. ACM SIGMOD*, 2002.

[24] J. Tayeb, O. Olusoy, and O. Wolfson. A Quadtree-Based Dynamic Attribute Indexing Method. *The Computer Journal*, 41(3):185–200, 1998.

[25] O. Wolfson, S. Chamberlain, S.Dao, L. Jiang, and G. Mendez. Cost and Imprecision in Modeling the Position of Moving Objects. In *Proc. ICDE*, pp. 588–596, 1998.

[26] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In *Proc. SSDBM*, pp. 111–122, 1998.