Indexing Mobile Objects on the Plane

Dimitris Papadopoulos
Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA, 92521
Tel: +1-909-787-2838
Fax: +1-909-787-4643
Email: dimitris@cs.ucr.edu

George Kollios
Department of Computer Science
Boston University
111 Cummington Street, Boston, MA, 02215
Tel: +1-617-358-1835
Fax: +1-617-353-6457
Email: gkollios@cs.bu.edu

Dimitrios Gunopulos
Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA 92521
Tel: +1-909-787-2479
Fax: +1-909-787-4643
Email: dg@cs.ucr.edu

Vassilis J. Tsotras
Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA 92521
Tel: +1-909-787-2888
Fax: +1-909-787-4643
Email: tsotras@cs.ucr.edu

Keywords: **Indexing Mobile Objects**

Abstract

We present a method to index objects moving on the plane in order to efficiently answer range queries about their future positions. This problem is motivated by real-life applications, such as predicting future congestion areas in a highway system, or allocating more bandwidth for areas where high concentration of mobile phones is imminent. We consider the problem in the external memory model of computation and present a dynamic technique. An experimental evaluation is included that shows the perfomance, scalability and efficiency of our method.

Παρουσιάζουμε μια μέθοδο για αναζήτηση αντικειμένων που κινούνται στο επίπεδο, έτσι ώστε να μπορούμε να απαντήσουμε ερωτήσεις σχετικά με τη μελλοντική τους θέση αποτελεσματικά. Το πρόβλημα παρουσιάζεται σε πολλές εφαρμογές, όπως για παράδειγμα πρόβλεψη κυκλοφοριακής συμφόρησης σε αυτικινητοδρόμους, ή αύξηση της χωρητικότητας σε δίκτυα κινητής τηλεφωνίας σε περιοχές όπου αναμένεται μεγάλη συγκέντρωση συνδρομητών. Χρησιμοποιούμε το μοντέλο υπολογισμού δευτερεύουσας μνήμης και παρουσιάζουμε μια δυναμική τεχνική. Παραθέτουμε επίσης πειραματική αξιολόγηση, η οποία δείχνει ότι η μέθοδός μας είναι πρακτική και εφαρμόσιμη.

Indexing Mobile Objects on the Plane

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 a method to index objects moving on the plane in order to efficiently answer range queries about their future positions. This problem is motivated by real-life applications, such as predicting future congestion areas in a highway system, or allocating more bandwidth for areas where high concentration of mobile phones is imminent. We consider the problem in the external memory model of computation and present a dynamic technique. An experimental evaluation is included that shows the perfomance, scalability and efficiency of our method.

1 Introduction

Location-aware applications, such as traffic monitoring, intelligent navigation, and mobile communications management, cannot be efficiently supported by traditional database management systems. The assumption that data stored in the database remain constant, unless explicitly updated, supports a model where updates are issued in discrete steps. The aforementioned applications deal with continuously changing data, i.e. objects' location. Inevitably, a DBMS receiving requests of that kind at every unit of time would exhibit tremendous update overhead.

* Supported by NSF CAREER Award 0133825.
† Supported by NSF CAREER Award 9984729, NSF IIS-9907477, and the DoD.
‡ Supported by NSF IIS-9907477 and the DoD.

An attractive solution to tackle the problem is to use a function of time $f(t)$ to abstract each object's location. Then the current location of a moving object at any time instant can be calculated. An update has to be issued only when the parameters of f change (e.g. speed or direction). Clearly, this approach minimizes the update overhead. However, it introduces new problems, such as the need for appropriate data models, query languages, and query processing and optimization techniques.

The focus of this paper is on the problem of indexing mobile objects in two dimensions. We are interested in efficiently answer range queries over the objects location in the future. Here we present techniques based on the duality transform to efficiently index the future locations of moving points on the plane. We extend the one dimensional techniques that were presented in [14]. We also present an experimental evaluation of our techniques.

In section 2 we give a formal problem description, and in section 3 we present work related with the problem at hand. Next, section 4 describes the dual transform, which is the core of our approach. The one-dimensional case is addresed in section 5, while the technique for indexing moving objects in 2-d is illustrated in section 6. Experimental results are given in section 7, and section 8 closes the paper.

2 Problem description

We consider a database that records the position of moving objects in two dimensions on a finite terrain. We assume that objects move with a constant velocity vector starting from a specific location at a specific time instant. Thus, we can calculate the future position of the object, provided that the characteristics of its motion remain the same. Velocities are bounded by $[v_{min}, v_{max}]$. Objects update their motion information, when their speed ot direction changes. The system is dynamic, i.e. objects may be deleted or new objects may be inserted.

Let $P(t_0) = [x_0, y_0]$ be the initial position at time t_0. Then, the object starts moving and at time $t > t_0$ its position will be $P(t) = [x(t), y(t)] = [x_0 + v_x(t - t_0), y_0 + v_y(t - t_0)]$, where $V = [v_x, v_y]$ is its velocity vector. For example, in Figure 1, the lines depict the objects' 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

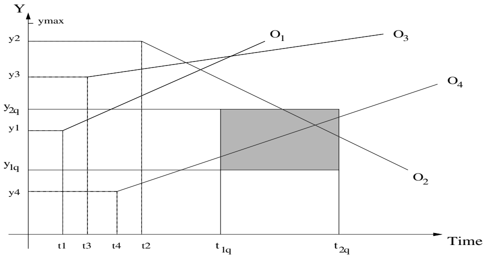

Figure 1: Trajectories and query in (t, y) plane.

$t_{now} \leq t_{1q} \leq t_{2q}$), given the current motion information of all objects" (i.e. the *two dimensional MOR query* in [14]).

3 Related work

The straightforward approach of representing an object moving on an 1-dimensional line is by plotting the trajectories as lines 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 (Figure 1). In this setting, 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 space-time approach provides an intuitive representation. Nevertheless, it is problematic, since the trajectories correspond to long lines. Using traditional indexing techniques in this setting tends to show many drawbacks.

One approach is to use a Spatial Access Method, such an R-tree [10] or an R*-tree [4]. In this setting 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. This could be accomplished by using an index such an R+-tree [19], a cell-tree [9], and the PMR-quadtree [18]. The shortcoming of these methods is that they introduce replication of the lines, since each line is copied into the cells that intersect it. Given the fact that lines are long, the situation becomes even worse. The effect of using space partitioning indices would be a high overhead for updates, along with large amount of space.

The work in [23] presents a technique to index mobile objects based on the PMR-quadtree. Also, the Moving Objects Spatio-Temporal (MOST) model and a language (FTL) for querying the current and future locations of mobile objects are presented in [20, 24, 25]. In order to index line segments a method based upon the dual transform was proposed in [11]. The use of dual transformation to index mobile objects is also proposed in [25].

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

Saltenis et al. [16] presented a technique to efficiently index moving objects. They proposed 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 moving object is represented by its location at a particular time instant (reference position) and its velocity vector. In [17] the R^{EXP}-tree was proposed to index moving objects with expiration time.

In [14] a technique to index moving objects was introduced, based upon the dual transform, which we extend here.

Related work also includes indexing animated objects [12, 15], nearest neighbor queries in a mobile environment [13, 8, 21], time-parameterized queries [22] and selectivity estimation for moving object queries [5].

4 The dual space-time representation

The dual transform, in general, maps a hyper-plane h from R^d to a point in R^d and vice-versa. In this section we briefly describe how we can address the problem at hand in a more intuitive way, by using the dual transform on the one-dimensional case.

Specifically, a line from the primal plane (t, y) is mapped to a point in the dual plane. A class of transforms with similar properties may be used for the mapping. The problem setting parameters determine which one is more usefull.

One dual transform for mapping the line with equation $y(t) = vt + a$ to a point in R^2 is by using the dual plane where one axis represents the slope of an object's trajectory (i.e. velocity) and the other axis its intercept. Thus we have the dual point (v, a) (this is called Hough-X transform in [11]). Accordingly, the 1-d query $[(y_{1q}, y_{2q}), (t_{1q}, t_{2q})]$ becomes a polygon in the dual space. By using a linear constraint query [7], the query Q in the dual Hough-X plane (Figure 2) is expressed in the following way [14]:

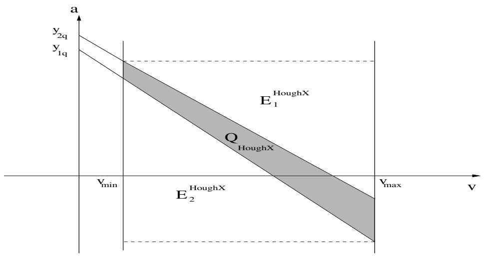

Figure 2: Query on the Hough-X dual plane.

- 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 $D_4 = a + t_{2q}v \leq y_{2q}$

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 point in the dual plane has coordinates (n, b), where $n = \frac{1}{v}$ and $b = -\frac{a}{v}$ (Hough-Y in [11]). Coordinate b is the point where the line intersects the line $y = 0$ in the primal space. By using this transform, horizontal lines cannot be represented. 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 are valid.

5 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 [14] an external memory technique was presented to tackle this problem, which is based upon partition trees. Lower bounds were given and it was shown that queries can be answered in $O(n^{\frac{1}{2}+\epsilon} + k)$ I/O's. Nevertheless, partition trees are not very useful in practice, because the hidden factor becomes large if a small ϵ is chosen.

In order to alleviate this property of partition trees, in the same work it was proposed to use a point access method in order to index the dual points. Even though these structures were designed to answer *orthogonal* range queries, Goldstein et al. [7] 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. By using this approach, in [14], the one dimensional MOR query is answered in the dual Hough-X space using logarithmic 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 $(t_{2q} - \frac{y_{1q}-y_r}{v_{min}}, \frac{1}{v_{min}})$.

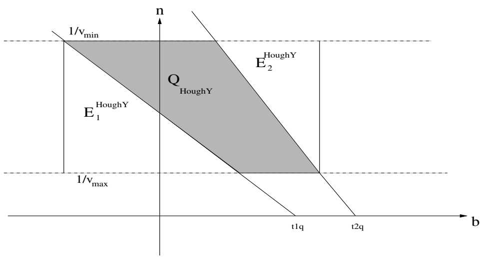

Figure 3: Query on the dual Hough-Y plane.

Since access methods are more efficient for rectangle queries, suppose that we approximate the simplex query with a rectangular one. In Figure 3 the query approximation 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 $E = E_1^{HoughY} + E_2^{HoughY}$ which is computed as:

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

We are interested in minimizing E since it represents a measure of the extra I/O's that an access method will have to perform for solving an one dimensional MOR query. E is based on both y_r (i.e. where the b coordinate is computed) and the query interval (y_{1q}, y_{2q}) which is unknown. Hence, we propose to keep κ indices (where κ is a small constant) at equidistant y_r's. All κ inidices 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 $y = \frac{y_{max}}{\kappa} \times i, i = 0, \ldots, \kappa - 1$. Conceptually, y_i serves as an "observation" element, and its corresponding index stores the data as observes from position y_i. A given one dimensional MOR query will be forwarded to the index(es) that minimize E. 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 κ "observation" indices can simply be a B+-tree [6]. In [14] it has been proved that the query can be answered with bounded E in logarithmic time.

6 Indexing in two dimensions

The 2-dimensional problem (Figure 4) is addresed 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. 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

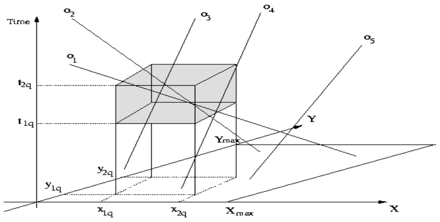

Figure 4: Trajectories and query in (x, y, t) space.

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 in 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 by 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

   - Partition the objects according to their velocity:

     (a) Objects with small velocity are stored using the Hough-X dual transform, while the rest are stored using the Hough-Y dual transform.

     (b) Motion information about the other projection is also included.

In order to choose one of the two projections and answer the simplex query, the technique descibed next is used.

6.2 Answering the query

The 2-d query is decomposed into two 1-d queries, one for each projection, and it is answered exactly. Furthermore, on a given projection, the simplex query is asked in both partitions, i.e. Hough-Y and Hough-X.

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 5, 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 c:

$$c = \frac{E^{HoughY}}{Q^{HoughY}} + \frac{E^{HoughX}}{Q^{HoughX}} \tag{2}$$

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-d query is presented next:

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 c for each projection, and choose the one (say p) that minimizes it.
4. Answer the query by searching the Hough-X and Hough-Y partition, using projection p.
5. Put an object in the result set, only if it satisfies the query. Use the whole motion information to do the filtering ”on the fly”.

7 Performance experiments

We present experimental results for the 2-dimensional MOR query, comparing our method and the TPR-tree. We used the same dataset generator in [16], as well as the same parameters suggested therein. The simulation scenario assumes objects moving on a finite terrain having size 1000 x 1000 km. The terrain contains a fully connected graph, whose edges are the routes objects can move along. The number of vertices, or destinations (ND), distinguish one dataset from another. The objects are initially positioned on the routes in a random fashion. They are assigned with equally probability to one of three possible groups having maximum velocity of 0.75 , 1.5 and 3km/min. Objects initially accelerate, then they maintain a maximum speed, and finally they decelerate. The updates are generated so that the total average time interval between two updates is fixed to a parameter UI. Queries consist of time-slice and window queries, and are issued within a time window W from the current time.

To test the performance of the two techniques we did the following experimental evaluation. First, we ran experiments were the TPR-tree has a fixed horizon $H = UI + W$ where $UI = 60$ and $W = 40$, and we used this parameters to generate the workload. We also generated queries that their temporal interval lies outside TPR-tree's predefined horizon (we added $5H$ to t_{1q} and t_{2q}). This situation can come up, if we are not able to accurately predict the horizon beforehand. Also this is similar to what happens if there are no updates for a while. In this case (i.e. no updates) the TPR-tree cannot answer queries efficiently, because it rebuilds the structure of the tree during updates only. This is necessary for the TPR-tree, because the size of the time-parameterized MBRs increases over time. For this second set of queries, we also tried the TPR-tree using the automatic horizon estimation [17].

We generated scenarios for 40 and 160 ND's. We also generated a dataset in which the objects can move randomly on the terrain. Each simulation scenario runs for 600 units and involves $100K$ objects. Four queries are issued every time instant, intermixed with around one million updates in total. The page size was set to $4K$ and a buffer pool of 50 pages was used. The leaf capacity was 204 in both methods. The update-time MBR setting was used for the TPR-tree.

The results of our experiments are shown next. Whenever shown in the figures, TPR-fixed stands for TPR-tree with fixed horizon setting, while TPR-auto stands for TPR-tree with automatic horizon estimation. In figure 5, the results regarding the queries with temporal part within the horizon are presented in terms of pages I/O. Similarly, in figure 6, the results for queries having temporal part outside the horizon are shown. Figure 7 presents the update performance results, while Figure 8 shows the space consumption. Note that, for our method, update performance and space consumption remain constant, regardless of whether the queries' temporal part lies within or outside TPR-tree's horizon.

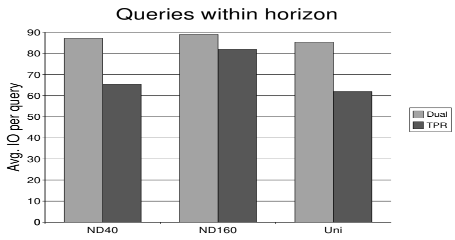

Figure 5: Queries within horizon

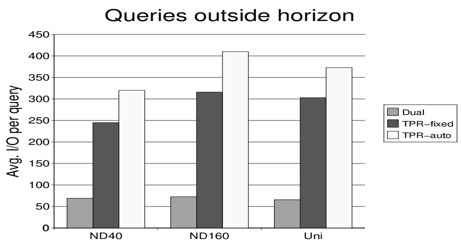

Figure 6: Queries outside of horizon

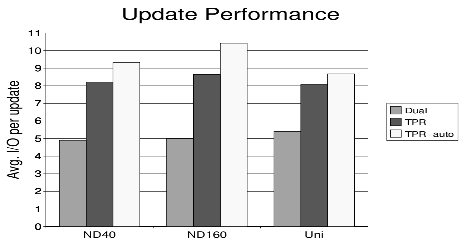

Figure 7: Update performance

We observe that both methods exhibit approximately equal performance for queries asked within the horizon. When the queries are issued outside the horizon, the TPR-tree perfomance is affected dramatically. Our method improves in this case, because the selectivity of the queries drops, when their temporal part lies far into the future. Furthermore, it has faster updates, but requires larger space. So there is a trade-off between update performance and space consumption. However, our method does not depend upon knowing the characteristics of the anticipated workload (i.e. the parameter

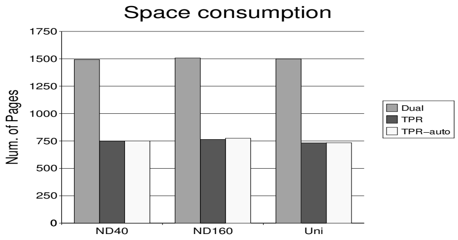

Figure 8: Space consumption

UI), and does not assume any query window W as the TPR-tree does.

8 Conclusions

We presented an external memory mechanism for indexing mobile objects that move on the plane, in order to efficiently answer range queries about their location in the future. The perfomance evaluation illustrates the applicability of our method.

AKNOWLEDGEMENTS

We would like to thank Simonas Saltenis for providing the source code for the TPR-tree.

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing Moving Points. *In Proc. of the 19th ACM Symp. on Principles of Database Systems (PODS)*, pages 175–186, 2000.

[2] L. Arge, V. Samoladas, and J.S. Vitter. On Two-Dimensinal Indexability and Optimal Range Search Indexing. In *Proc. of the 18th ACM Symp. on Principles of Database Systems (PODS)*, pages 346–357, June 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*, pages 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. of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, May 1998.

[5] Y.-J. Choi and C.-W. Chung. Selectivity Estimation for Spatio-Temporal Queries to Moving Objects. In *Proc. of the ACM SIGMOD*, Madison, Wisconsin, June 2002.

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

[7] J. Goldstein, R. Ramakrishnan, U. Shaft, and J.B. Yu. Processing Queries By Linear Constraints. In *Proc. 16th ACM PODS Symposium on Principles of Database Systems*, pages 257–267, Tuscon, Arizona, 1997.

[8] 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.

[9] O. Gunther. The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases. In *Proc. 5th IEEE Inter. Conf. on Data Engineering*, Los Angeles, CA, USA, February 1989.

[10] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proc. ACM SIGMOD*, pages 47–57, Boston, Mass, June 1984.

[11] H. V. Jagadish. On Indexing Line Segments. In *Proc. 16th. International Conference on Very Large Data Bases*, pages 614–625, Brisbane, Queensland, Australia, August 1990.

[12] G. Kollios, D. Gunopulos, and V. Tsotras. Indexing Animated Objects. *In Proc. 5th Int. MIS Workshop, Palm Springs Desert, CA*, 1999.

[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*, pages 119–134, 1999.

[14] G. Kollios, D. Gunopulos, and V. Tsotras. On Indexing Mobile Objects. *In Proc. of the 18th ACM Symp. on Principles of Database Systems (PODS)*, pages 261–272, June 1999.

[15] G. Kollios, V. J. Tsotras, D. Gunopulos, A. Delis, and M. Hadjieleftheriou. Indexing Animated Objects Using Spatiotemporal Access Methods. *TKDE*, 13(5):758–777, 2001.

[16] S. Saltenis, C. Jensen, S. Leutenegger, and Mario A. Lopez. Indexing the Positions of Continuously Moving Objects. *In Proceedings of the ACM SIGMOD*, pages 331–342, May 2000.

[17] S. Saltenis and C. S. Jensen. Indexing of Moving Objects for Location-Based Services. In *Proc. 18th. Inter. Conference on Data Engineering*, San Jose, CA, Feb 2002.

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

[19] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proc. 13th Inter. Conf. on Very Large Data Bases*, pages 507–518, Brighton, England, September 1987.

[20] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. *In Proceedings of the 13th ICDE, Birmingham, U.K*, pages 422–432, April 1997.

[21] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *Proc. SSTD 2001*, pages 79–96, Redondo Beach, CA, July 2001.

[22] Y. Tao and D. Papadias. Time-Parameterized Queries in Spatio-Temporal Databases. In *Proc. of ACM SIGMOD*, Madison, Wisconsin, June 2002.

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

[24] O. Wolfson, S. Chamberlain, S.Dao, L. Jiang, and G. Mendez. Cost and Imprecision in Modeling the Position of Moving Objects. In *Proc. 14th IEEE Inter. Conf. on Data Engineering*, pages 588–596, Orlando, Florida, February 1998.

[25] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In *Proc. of 11th Int. Conf. on Scientfic and Statistical Database Management*, pages 111–122, Capri, Italy, Jul 1998.