INDEXING OF MOVING OBJECTS

Petr Částek
Doctoral Degree Programme (1), FIT BUT
E-mail: xcaste01@stud.fit.vutbr.cz

Supervised by: Jaroslav Zendulka
E-mail: zendulka@fit.vutbr.cz

ABSTRACT

With the recent advances in wireless networks, embedded systems, and GPS technology, databases that manage the location of moving objects have received increased interest. In particular, we propose methods to index moving objects in order to efficiently answer range queries about their current and future positions. We address the problem in external memory and present dynamic solutions, both for the one-dimensional and the two-dimensional cases. Our approach transforms the problem into a dual space that is easier to index.

1. INTRODUCTION

A spatiotemporal database system manages data whose geometry changes over time. There are many applications that create such data including global change (as in climate or land cover changes), transportation (traffic surveillance data, intelligent transportation systems), social (demographic, health,etc.), and multimedia (animated movies) applications. In general, one could consider two spatial attributes of spatiotemporal objects that are time dependent, namely, position (i.e., the object's location inside some reference space) and extent (i.e., the area or volume the object occupies in the reference space).

The usual assumption in traditional database management systems is that data stored in the database remain constant until explicitly changed by an update. This model is appropriate when data change in discrete steps, but it is inefficient for applications with continuously changing data. A better approach is to abstract each object's location as a function of time *f(t)* and update the database only when the parameters of *f* change (for example when the speed or the direction of a car changes). Using *f(t)* the "motion" database can compute the location of the mobile object at any time in the future. While this approach minimizes the update overhead, it introduces a variety of novel problems (such as the need for appropriate data models, query languages, and query-processing and optimization techniques) since the database is storing not data values but rather functions to compute these values [1].

2. ANALYSIS

These motion database problems have recently attracted the interest of the research community. Sistla et al. and Wolfson et al. [1] present the Moving Objects Spatio-Temporal (MOST) model and a language (FTL) for querying the current and future locations of mobile objects; Guting et al. propose a model that tracks and queries the history (past routes) of mobile objects based on new spatiotemporal data types. Another spatiotemporal model appears in [1]. Spatiotemporal queries about mobile objects have important applications in traffic monitoring, intelligent navigation or mobile communication. For example, in mobile communication systems, we could allocate more bandwidth in areas where a high concentration of mobile phones is approaching.

Another important issue in spatiotemporal databases is related to the protection of the privacy of mobile users. Recent directives and regulations, such as the European directive 58/2002/EC, specify that the location information of mobile users constitutes sensitive private information and must be protected against unauthorized use. Another approach is to allow only aggregate queries (COUNT, SUM, AVG) that do not reveal object IDs [1].

2.1. The dual space-time representation

In general, the dual transformation is a method that maps a hyperplane 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 for 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 useful. One dual transform for mapping the line with equation $y(t) = vt + a$ to a point in R^2 is to consider the dual plane where one axis represents the slope of an object's trajectory (velocity) and the other axis its intercept (Fig. 1). Thus we get the dual point *(v,a)* (this is called Hough-X transform in [1]). Similarly, a point $p = (t,y)$ in the primal space is mapped to line $a(v) = -tv+y$ in the dual space. An important property of the duality transform is that it preserves the above-below relationship. As is shown in Fig. 1, the dual line of point *p* is above the dual point l^* of line *l*.

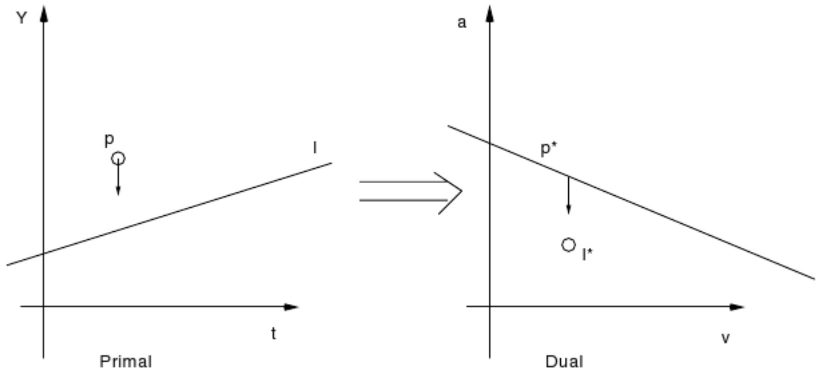

Fig. 1.: Hough-X dual transformation: primal plane *(left)*, dual plane *(right).*

Based on the above property, it is easy to show that the one-dimensional query [(y_{1q} , y_{2q}), (t_{1q} , t_{2q})] becomes a polygon in the dual space. Consider a point moving with positive velocity. Then the trajectory of this point intersects the query if and only if it intersects the segment defined by the points p_1 = (t_{1q} , y_{2q}) and p_2 = (t_{2q} , y_{1q}). Thus, the dual point of the trajectory must be above the dual line p_2^* and below p_1^* . The same idea is used for the negative velocities. Therefore, using a linear constraint query, the query Q in the dual Hough-X plane is expressed in the following way:

- If $v > 0$ then $Q = C_1 \wedge C_2$ where $C_1 = a + t_{2_q} v \geq y_{1_q}$ and $C_2 = a + t_{1_q} v \leq y_{2_q}$.
- If $v < 0$ then $Q = D_1 \wedge D_2$ where $D_1 = a + t_{1_q} v \geq y_{1_q}$ and $D_2 = a + t_{2_q} v \leq y_{2_q}$.

By rewriting the equation $y(t) = vt + a$ as $t = 1/v\, y - A/V$, we can arrive at a different dual representation. Now the point in the dual plane has coordinates *(b,n)*, where $b = -A/V$ and $n = 1/V$ (Hough-Y). Coordinate *b* is the point where the line intersects 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. Therefore, for static objects, we can use only the Hough-X transform [1].

3. INDEXING IN ONE DIMENSION

In this section we illustrate techniques for the one-dimensional case, i.e., for objects moving on a line segment. There are various reasons for examining the one-dimensional case. First, the problem is simpler and can give good intuition about the various solutions. It is also easier to prove lower bounds and approach optimal solutions for this case. Moreover, it can have practical uses as well. A large highway system can be approximated as a collection of smaller line segments (this is the 1.5-dimensional problem) [1].

3.1. A lower bound

By using the dual space-time representation, the problem of indexing moving 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 points in two dimensions, and we want to answer efficiently queries of the following form: *"Given a set of linear constraints ax ≤ b, find all points in S that satisfy all the constraints."* Geometrically, we want to find the points in the interior of the polygon. They show that simplex reporting in *d*-dimensions with a query time of $O(N^{\delta} + K)$, , where *N* is the number of points, *K* is the number of reported points, and $0 < \delta \leq 1$, requires space $\Omega(N^{d(1-\delta)-\varepsilon})$ for any fixed .

The bound holds for the static case, even if the query region is the intersection of just two hyperplanes. Since can be arbitrarily small, any algorithm that uses linear space for *d*-dimensional range searching has a worst-case query time of $O(N^{(d-1)/d} + K)$. . Here we show that a similar bound holds for the I/O complexity of simplex searching. We use the external memory pointer machine as our model of computation. This is a generalization of the pointer machine suitable for analyzing external memory algorithms. In this model, a data structure is modeled as a directed graph *G = (V,E)* with a source *w*. Each node of the graph represents a disk block and is therefore allowed to have *B* data and pointer fields. The points are stored in the nodes of G. Given a query, the algorithm traverses *G* starting from *w*, examining the points at the nodes it visits. The algorithm can only visit nodes that are neighbors of already visited nodes (with the exception of the root) and, when it terminates the answer to the query, must be contained in the set of visited nodes. The running time of the algorithm is the number of nodes it visits.

3.2. Optimal solution

Matousek [3] gave a near optimal algorithm for simplex range searching, given a static set of points. This *main-memory* algorithm is based on the idea of simplicial partitions. We briefly describe this approach. For a set *S* of *N* points, a simplicial partition of *S* is a set $\{(S_1, \Delta_1) \ldots (S_r, \Delta_r)\}$, where $\{S_1, \ldots, S_r\}$ is a partitioning of *S* and Δ_i is a triangle that contains all the points in S_i. If $max_i|S_i| < 2min_i|S_i|$, where $|S_i|$ is the cardinality of the set S_i, we say that the partition is balanced. Matousek shows that, given a set *S* of *N* points and a parameter s (where $0 < s < N/2$), we can construct, in linear time, a balanced simplicial partition for *S* of size *O(s)* such that any line crosses at most *O(s)* triangles in the partition.

This construction can be used recursively to construct a partition tree for *S*. The root of the tree contains the whole set *S* and a triangle that contains all the points. We find a balanced simplicial partition of *S* of size $\sqrt{|S|}$. Each of the children of the root are associated with a set S_i from the simplicial partition and the triangle Δ_i that contains the points in S_i. For each of the S_i's we find simplicial partitions of size $\sqrt{|S|}$ and continue until each leaf contains a constant number of points. The construction time is $O(N \log_2 N)$ [1].

3.3. 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. An outline of the procedure for building the index follows:

1. Decompose the two-dimensional motion into two one dimensional 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 $|v| < VT$ are stored using the Hough-X dual transform, while objects with $|v| \geq VT$ are stored using the Hough-Y dual transform.
     (b) Motion information about the other projection is also included in each point.

In order to choose one of the two projections and answer the simplex query [1].

4. B^{DUAL}-TREE

A B^{dual}-tree has two parameters: a *horizon H*, and a *reference time* T^{ref}. *H* decides the farthest future time that can be efficiently queried. Similar to TPR-trees [1], a B^{dual}-tree constructed at time t optimizes queries about the period *[t,t+H]*. Queries that concern timestamps later than $t + H$ are also correctly answered, but they are not optimized due to their lower importance (predicting about a distant timestamp is not useful since many objects may have issued updates by then). The second parameter T^{ref} is needed to convert data to their duals. The T^{ref} is not necessarily equal to the construction time of the tree.

B^{dual}-tree is composed of two B^+-trees, *BT1* and *BT2* . Each tree has two states:

- a growing state when objects can be inserted/deleted
- a shrinking state when only deletions are allowed

At any time, one tree is in the growing state, and the other in the shrinking state. Initially, *BT1* (*BT2*) is in the growing (shrinking) state for time interval *[0,T)*, when all the updates are directed to *BT1* , and *BT2* remains empty. During *[T,2T)*, the states of *BT1* and *BT2* are reversed. In this period, every insertion is performed in *BT2*. A deletion, however, may remove an object from *BT1* or *BT2* , depending on whether it was inserted during *[0,T)* or *[T,2T)*, respectively. At time *2T*, *BT1* becomes empty (all the objects inserted during *[0,T)* have issued updates), and the two trees switch states again. Given a query, both BT1 and BT2 are searched, and the results are combined to produce the final answer [2].

5. CONCLUSIONS

We presented external memory techniques for indexing moving objects in order to efficiently answer range queries about their location in the future. By employing dual transformations we illustrated efficient indexing schemes for the one-dimensional (moving on a line) as well as the two-dimensional case. less update cost. Updating is an important consideration given the highly dynamic environment of moving objects. Moreover, the duality approach does not require the specification of a predefined horizon. An interesting direction of future research is joins among relations of mobile objects. We proposed the B^{dual}-tree, a new spatiotemporal index for predictive search that combines features and advantages of *state-of-the-art* methods. Furthermore, it would be worth considering the problem in the context of uncertainty in the position and velocity of the mobile objects. The relationship of indexing techniques and protection of privacy of mobile users is also a very interesting problem that we plan to consider.

REFERENCES

[1] KOLLIOS, George, PAPADOPOULOS, Dimitris, GUNOPULOS, Dimitrios. Indexing mobile objects using dual transformations. The VLDB Journal : Digital Object Identifier (DOI). 2005, no. 14, s. 238-256.

[2] YIU, Man Lung, TAO, Yufei, MAMOULIS, Nikos. The B^{dual}-tree: indexing moving objects by space filling curves in the dual space . The VLDB Journal. 2008, no. 17, s. 379-400.

[3] MATOUŠEK, J. Efficient partition trees. Discrete Computer Geometrics . 1992, č. 8, s. 432-448. Available from www: <http://www.mpi-inf.mpg.de/~sgovinda/Course/papers/M92.pdf>.

[4] ARGE, Lars, SAMOLADAS, Vasilis, VITTER, Jeffrey. On two-dimensional indexability and optimal range search indexing. Proceedings of the 18th ACM PODS. 1999, s. 346-357. Available from www: <http://www.daimi.au.dk/~large/Papers/rangepods99.ps>.