A Redundant Bi-Dimensional Indexing Scheme for Three-Dimensional Trajectories

Antonio d'Acierno
Institute of Food Science
National Research Council
Avellino, Italy
dacierno.a@isa.cnr.it

Alessia Saggese, Mario Vento
DIEII
Universitá degli Studi di Salerno
Fisciano (SA), Italy
{*asaggese,mvento*}*@unisa.it*

***Abstract*—The need of efficient methods for querying continuously moving object databases arises in many applications of intelligent video surveillance. As a consequence, several data indexing strategies have been introduced in order to improve data storing and retrieving and develop more efficient trajectory analysis systems. However, even though efficient spatial indexes in bi-dimensional planes are usually available, several issues occur when data to be handled are three- or even four-dimensional as, for instance, moving objects trajectories in real world environments. For this reason, we are interested in proposing a new indexing scheme capable of analysing and retrieving three-dimensional trajectories in efficient way. This goal is achieved by redundantly projecting and analysing a collection of trajectories on bi-dimensional planes and validating the obtained result through a clipping algorithm. Experimental results show that the proposed approach yields good performance in terms of averaged retrieving time when applied to time interval queries.**

Keywords-MOD; Three-dimensional trajectory; Indexing; Time interval query.

I. INTRODUCTION

Moving Object Databases (MODs) are used to store continuously moving objects. According to the widely adopted line segments model [1], the object motion is expressed through its trajectory; trajectories, in turn, are represented by a polyline in a three-dimensional space, the first two dimensions being referred to space and the third one to time (Figure 1).

The demand of efficiently querying MODs arises in many contexts, from air traffic control to mobile communication systems. There are at least two categories of queries that are worth to be considered: queries about the future positions of objects, and queries about the historical positions of moving objects. Historical queries can be further classified [1] into coordinate-based queries and trajectory based queries. While trajectory-based queries involve information about a trajectory such as topology and velocity, coordinate-based queries in turn include:

1) Time interval queries: select all objects within a given area and within a given time period;
2) Time-slice queries: select all the objects present in a given area at a given time instant;
3) Nearest neighbor queries: select the k nearest neighbor objects to a given point in space at a given time instant.

A key problem to be addressed concerns the indexing of these data. R-trees, proposed by Guttman in his pioneering paper [2], was a widely adopted solution motivated by the Very Large Scale Integration (VLSI) design: how to efficiently answer whether an area is already covered by a chip. R-trees hierarchically organize geometric objects representing them using the MBRs (*Minimum Bounding Rectangles*); each internal node corresponds to the MBR that bounds its children while a leave contains pointers to objects. Starting from the original structure, several optimizations have been proposed [3]; in [1], for example, the particularities of spatio-temporal data are captured by two access methods (STR-tree and TB-tree) while SEB-trees [4] segment space and time.

When the aim is to index and query repositories of large trajectories, the size of MBRs can be reduced segmenting each trajectory and then indexing each sub-trajectory using R-Trees; such an approach is described, for example, in [5], where a dynamic programming algorithm to minimize the I/O for an average size query is proposed. SETI [6] segments trajectories and groups sub-trajectories into a collection of *spatial partitions*; queries run over the partitions that are most relevant for the query itself. TrajStore [7] co-locates on a disk block (or in a collection of near blocks) trajectory segments using an adaptive multi-level grid; thanks to this method, it is possible to answer a query only reading a few blocks.

Our main aim is to extend a video surveillance system [8] with an efficient method for querying continuously moving object databases in order to interpret the behaviour of different entities populating a scene. Even though efficient bi-dimensional indexing methods are usually available, several problems arise when data to be handled are three- or even four-dimensional as happens for the considered video surveillance system. Indeed, this framework identifies a real object by using a triple (x, y, f) where (x, y) represents the

 ISBN: 978-1-61208-162-5

object position whereas f is the frame number; assuming a constant frame rate, the frame number and the time can be used as synonyms. In order to achieve this aim, we extend the existing video surveillance system by proposing a new indexing scheme capable of analysing and retrieving three-dimensional trajectories in efficient way. The proposed method works by redundantly projecting and analysing a collection of trajectories on bi-dimensional planes. Obtained result is finally validated in the three-dimensional plane through a clipping algorithm. Different experiments, performed by using the POSTGIS [9] system, will show that the proposed approach yields good performance in terms of averaged retrieving time when applied to time-interval queries on synthetic data.

II. The Proposed Solution

A trajectory is usually referred to as a list of space-time points:

$$< (x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_N, y_N, t_N) >$$

where the generic pair (x_i, y_i) is the spatial location and t_i represents the time. Each point is thus treated as an object in an extended spatial domain, since time is considered as an additional dimension. As already mentioned, we use the line segments model [1], each segment being the linear interpolant between two consecutive points.

To answer a time-interval query, we have to verify the intersection between a 3D query box, identified by bottom-left-back $(x_{min}, y_{min}, t_{min})$ and top-right-front $(x_{max}, y_{max}, t_{max})$ points, and all the segments of each trajectory. To determine if a line segment lies inside, outside or partially outside the box, we can use a clipping algorithm; one of the most efficient methods for our purposes is the extension to 3D of the 2D Cohen-Sutherland Line Clipping Algorithm [10].

The recursive bi-dimensional Cohen-Sutherland Line Clipping Algorithm considers only segment endpoints; if at least one endpoint of the segment s lies inside the clip box, the hypothesis *h: s intersects the box* can be trivially accepted. If both endpoints are outside the clip box, the segment may or may not intersect with the clip box. In some cases *h* can be still trivially accepted (as it happens for segments AB in Figure 2) or rejected (segment CD). Other situations (segment EF and segment GH) can be solved recursively by subdividing the line into two segments and using the extensions of the clip box edge; one of the obtained segment can be trivially rejected, while the other one is the new segment to be analyzed.

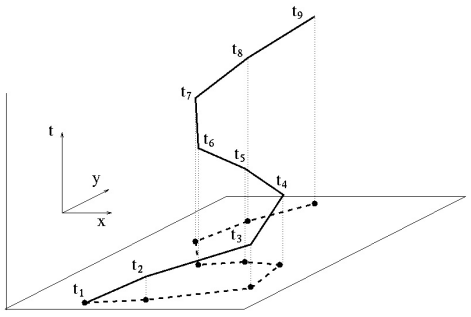

Figure 1. An example of spatio-temporal trajectory; x and y dimensions refer to position while the third dimension (t) refers to time.

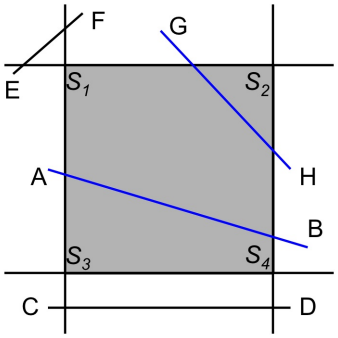

Figure 2. Some segments lying inside (AB and GH) or outside (CD and EF) the clipping area $S_1S_2S_3S_4$.

The bi-dimensional Cohen-Sutherland algorithm can be easily extended to the 3D case [10]; here, operations have to be performed with reference to six half-planes ($y < y_{min}$, $y > y_{max}$, $x < x_{min}$, $x > x_{max}$, $t < t_{min}$, $t > t_{max}$) and by considering the obtained twenty-seven regions.

In the worst case, when the trajectory does not intersect the box, we have to verify all the segments in the trajectory; such an approach is too expensive for a large amount of trajectory data, thus the aim of the proposed indexing strategy is to reduce the number of candidate trajectories to clipping, taking advantage of the existing 2D indexes.

The method we propose is based on three derived bi-dimensional spaces obtained by projecting each 3D trajectory onto (X, Y), (X, T) and (Y, T) planes. It is worth to observe that if a trajectory intersects the 3D query box, then each trajectory projection will intersect the correspondent query box projection. This is a necessary but not sufficient condition since the opposite is clearly not true: if all projections trajectory intersect correspondent box projection on considered spaces, they do not have to intersect the 3D query box too. To better explain this concept, Figure 3 shows a trajectory on 3D space and its projections on 2D spaces: we can note that all the trajectory projections intersect correspondent box projection, although the trajectory does not intersect the 3D query box.

Figure 4 resumes the main phases of the method needed to answer a time-interval query. For each three-dimensional trajectory t (Figure 4a), we redundantly store three bi-dimensional trajectories. Each trajectory is obtained by projecting t on the XY plane (t_{XY}), on the XT plane (t_{XT})

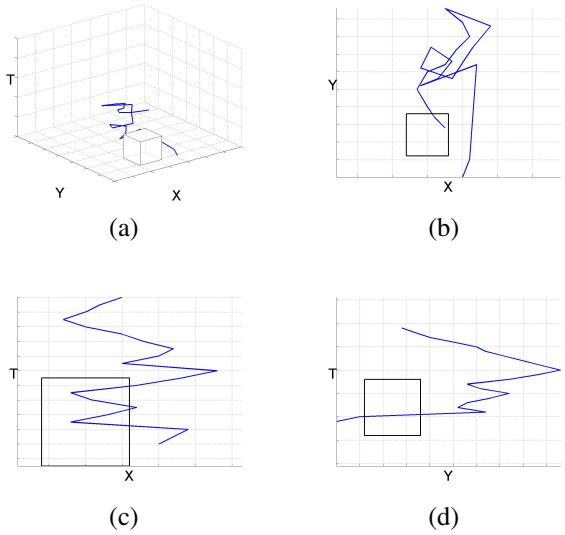

Figure 3. An example of trajectory (a) and its projections on the different coordinate planes XY (b), XT (c) and YT (d). Note that although the trajectory does not intersect the query box, its projections do it.

and on the YT plane (t_{YT}), as shown in Figure 4b. Given a box B representing the time-interval query to be solved, we similarly consider B_{XY}, B_{XT} and B_{YT}.

Using one of the available bi-dimensional indexes, we can find on each plane the following three trajectory sets in a very simple and efficient manner (Figure 4c):

$$T_{XY} = \{t_{XY} : MBR(t_{XY}) \cap B_{XY} \neq \emptyset\} \quad (1)$$

$$T_{XT} = \{t_{XT} : MBR(t_{XT}) \cap B_{XT} \neq \emptyset\} \quad (2)$$

$$T_{YT} = \{t_{YT} : MBR(t_{YT}) \cap B_{YT} \neq \emptyset\} \quad (3)$$

The set T of the candidate trajectories to be clipped in 3D space is thus trivially defined as:

$$T = \{t : t_{XY} \in T_{XY} \wedge t_{XT} \in T_{XT} \wedge t_{YT} \in T_{YT}\} \quad (4)$$

As shown in Figure 4d, the candidate set is composed by trajectories whose MBR on each plane intersects the corresponding projection of the query box; this does not imply that, for example, $t^i_{XY} \in T_{XY}$ actually intersects B_{XY}. This choice will be motivated in the last Section.

III. Experimental Results

We test our system over synthetic data sets generated as follows.

Let W and H be the width and the height of our scene; let S be the time interval we are interested in. Each trajectory starting point is randomly chosen in our scene at a random time instant t_1; the trajectory length is assumed to follow a Gaussian distribution. We also randomly chose an initial direction along the x axis (d_x) as well as a direction along the y axis (d_y).

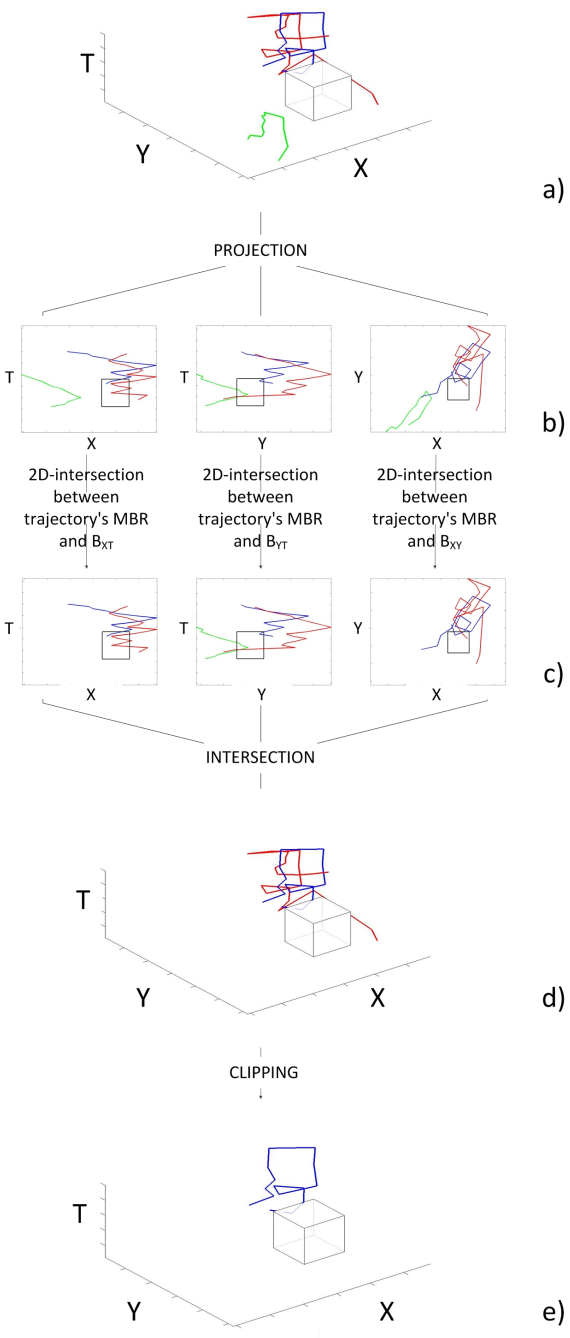

Figure 4. An overview of the proposed method. Figure a) shows a query box and some examples of trajectories; in b) there are the projections of each trajectory on the coordinate planes. Figure c) only shows the projections that intersect the correspondent query box. Figure d) shows the trajectories whose three projections intersect correspondent boxes. Finally, Figure e) shows the final result of our method, i.e., trajectories that really intersect the query box.

Table I
THE PARAMETERS USED TO GENERATE OUR DATA.

Scene width	W	10^3
Scene height	X	10^3
Time interval length in seconds	S	10^4
Number of trajectories (thousands)	T	{1,2,3,5,10}
Mean number of points in each trajectory (thousands)	$\overline{L}$	{1,2,3,4,5,10}
Standard deviation for trajectory's length	σ_L	10
Probability to invert the direction along x	PI_x	5%
Probability to invert the direction along y	PI_y	5%
Maximum velocity along x (pixels/seconds)	V_x^{max}	10
Maximum velocity along y (pixels/seconds)	V_y^{max}	10

At each time step t, we first generate the new direction, assuming that d_x (d_y) can be changed with probability PI_x (PI_y) and then we randomly chose the velocity along x and y (expressed in pixels/seconds and assumed to be greater than 0 and less than two fixed maxima). The new position of the object can be so easily evaluated; if it does not belong to our scene, d_x and/or d_y are changed.

We define the scene populated with trajectories as the "Scenario". Table I reports the free parameters to be chosen together with the values we chose to create the 30 different scenarios used in our experiments.

We store data in Postgres using PostGIS, an extension built to store and query spatial data like points, lines and polygons. We represent mobile object trajectory as a tuple of $(mId, mTraj_{XY}, mTraj_{XT}, mTraj_{YT})$, where mId is the unique trajectory identifier and $mTraj_{XY}$ (respectively $mTraj_{XT}$ and $mTraj_{YT}$) is the XY projection of the trajectory (respectively, the XT and the YT projections), represented as a sequence of segments (a PostGIS multi-line). Data are indexed using the R-tree over GIST (Generalized Search Trees) indexes [11] since it guarantees, compared with the PostGIS implementation of R-trees, best performances for spatial queries. Similarly to R-trees, GIST indexes break up data into a search tree according to their spatial position. Once data have been indexed, PostGIS provides a very efficient function to perform intersection between boxes and MBRs in a 2D space. It is clear that this kind of intersection could be not accurate, especially for large trajectories whose MBR, especially in the XY plane, could cover almost the entire area.

To test our system we need some queries, each query being defined by the corresponding three dimensional cube. The dimension D_c and the position P_c of the cube of course affect the obtained performance. We decided thus to test different dimensions, expressed as a percentage of the whole volume; we choose $D_c \in \{1\%, 5\%, 10\%, 20\%, 30\%, 50\%\}$. Each query is repeated several times (to be more precise, smaller cubes are queried more times) and results are then averaged.

The overall averaged querying time ($\overline{QT}$) of course depends on T, on $\overline{L}$ and on D_c; $\overline{QT}$ can be expressed as the sum of two terms: $\overline{QT_q}$ is the time needed to extract data from the database while $\overline{QT_c}$ is the time needed to apply the clipping algorithm to candidate trajectories.

We conduct our experiments on a PC equipped with an Intel quad core CPU at 2.66 GHz, using the 32 bit version of the PostrgreSQL 9.0 server and the 1.5 version of PostGIS. We obtain that, on average, $\frac{\overline{QT_c}}{\overline{QT_c}+\overline{QT_q}} = 50.4\%$. In the following we do not further investigate on $\overline{QT_c}$ and $\overline{QT_q}$ but we will concentrate on how $\overline{QT}$ increases as the free parameters vary.

Diamonds in Figure 5 express (in a log-log scale for the sake of readability) $\overline{QT}$ in seconds as the number of trajectories varies for different values of $\overline{L}$, both for small cubes ($D_c = 1\%$) and for large ones ($D_c = 30\%$). To analyze the relationship between $\overline{QT}$ and T we polynomially approximate $QT(T)$, both for each fixed D_c and for each $\overline{L}$. We obtain, with a very good approximation, that $\overline{QT}$ linearly increases with T (lines in Figure 5).

Diamonds in Figure 6 express $\overline{QT}$ in seconds as the dimensions of the cubes vary, having $\overline{L}$ as parameter and for several values of T. In this case we obtain that $\overline{QT}$ quadratically depends on D_c (lines in Figure 6).

Last diamonds in Figure 7 express $\overline{QT}$ in seconds as $\overline{L}$ varies for several values of D_c; in this case we have again a quadratic dependency on $\overline{L}$.

IV. CONCLUSIONS AND FUTURE DIRECTIONS

In the framework of a video surveillance system, we are interested in efficiently querying a three dimensional MOD using off the shelf solutions and so, in this paper, we propose a redundant storing system to index large repositories of three dimensional trajectories using widely available two dimensional indexes; the proposed method has been implemented using PostGIS, the well known spatial extension of the PostgreSQL server. Preliminary results, obtained on time interval queries performed against synthetic data, show that the proposed solution is able to fully exploit retrieving capabilities based on well established two dimensional indexes.

Concerning our work in progress, it must be observed that there are several possibilities to improve the performance of our system. First, we have a querying time that linearly increases with T while is a quadratic function of $\overline{L}$; thus, it can be pointed out that it is better to have more trajectories

 ISBN: 978-1-61208-162-5

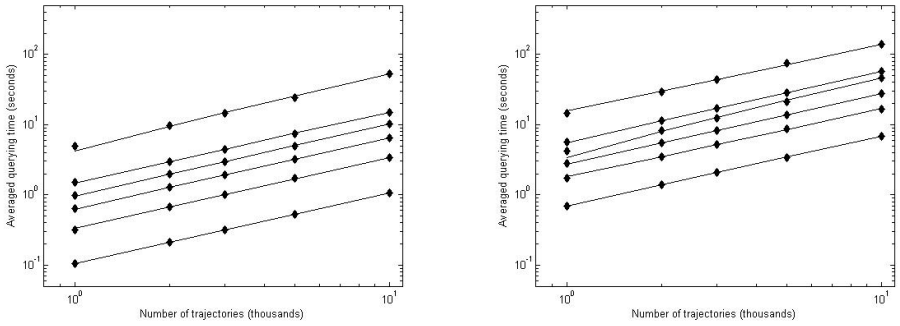

Figure 5. $\overline{QT}$ (in seconds) as the number of trajectories increases having the number of points in each trajectory (in thousands) as parameter.

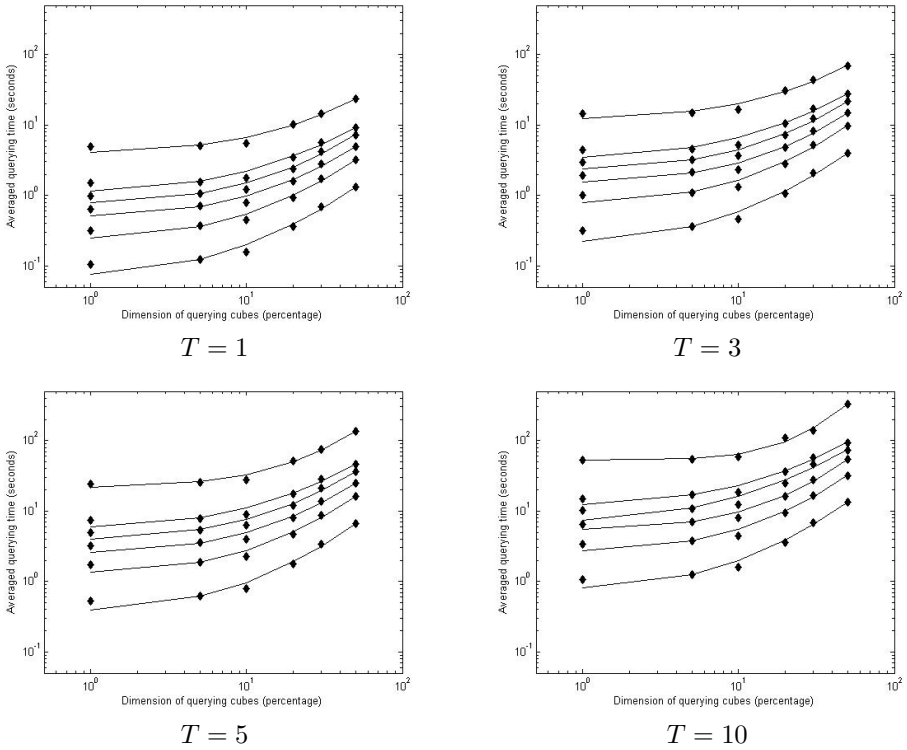

Figure 6. $\overline{QT}$ (in seconds) as the dimension of the querying cube (in percentage of the whole volume) increases and having $\overline{L}$ as parameter.

with fewer line segments and this can be obtained pursuing at least two strategies. A trajectory can be easily *compressed* because, in many context, data are highly redundant when sampling objects' positions at high rate. A trajectory can be then splitted in two or more sub-trajectories and applying our method to the set of sub-trajectories; such an approach, while clearly improves the performance due to the linear dependency of $\overline{QT}$ on T, also optimizes MBR-based indexes.

It is then worth to be noted that the clipping algorithm has to be applied in parallel to each candidate trajectory. This step can be easily implemented using multi threading, in order to take advantage from multi-core and multi-processors systems. On the other hand, the functions testing if two geometries intersect, available in GIS systems, are typically applied sequentially on each considered pair. For such a reason we query our DB (on each projected plane) for trajectories whose MBR intersects the corresponding projection of the query box (a very fast query given the MBR based index), instead of trajectories that actually intersect the corresponding MBR, that have to be tested sequentially.

Furthermore, we store data redundantly since, for each trajectory t, we store t_{XY}, t_{XT} and t_{YT} so that our

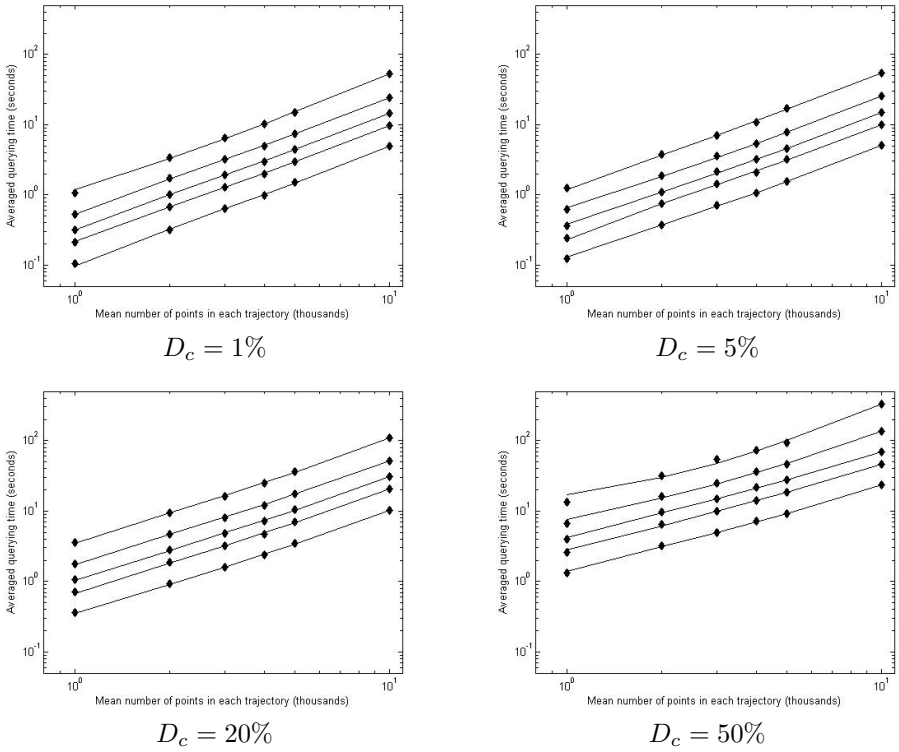

Figure 7. $\overline{QT}$ (in seconds) as the number of points in each trajectory (in thousands) increases and having the number of trajectories as parameter.

schema roughly doubles the used memory (for each three dimensional point we store three bi-dimensional points) and this can easily become a serious limitation when the number of stored trajectories increases. To overcome such a problem we are developing a new schema that heavily diminishes data redundancy.

The system then needs to be extended; in fact queries different from the time-interval ones are likely to be easily solvable with our solution.

Last, another objective is to make our system able to store and handle data as they are acquired.

REFERENCES

[1] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches in query processing for moving object trajectories," in *Proceedings of VLDB Conference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 395–406.

[2] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings ACM SIGMOD Conference*. New York, NY, USA: ACM, 1984, pp. 47–57.

[3] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, *R-Trees: Theory and Applications*. Springer, 2005.

[4] Z. Song and N. Roussopoulos, "Seb-tree: An approach to index continuously moving objects," in *Proceedings of the 4th Conference on MDM*. London, UK, UK: Springer-Verlag, 2003.

[5] S. Rasetic, J. Sander, J. Elding, and M. A. Nascimento, "A trajectory splitting model for efficient spatio-temporal indexing," in *Proceedings of the 31st international Conference on VLDB*. VLDB Endowment, 2005, pp. 934–945.

[6] V. P. Chakka, A. Everspaugh, and J. M. Patel, "Indexing large trajectory data sets with seti," in *CIDR*, 2003.

[7] P. Cudre-Mauroux, E. Wu, and S. Madden, "Trajstore: An adaptive storage system for very large trajectory data sets," *Data Engineering, International Conference on*, vol. 0, pp. 109–120, 2010.

[8] D. Conte, P. Foggia, G. Percannella, and M. Vento, "Performance evaluation of a people tracking system on pets2009 database," in *Proceedings of the 7th IEEE International Conference on AVSS*, 2010, pp. 119–126.

[9] "POSTGIS," `http://postgis.refractions.net//`.

[10] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice in C (2nd Edition)*. Addison-Wesley, 2004.

[11] J. M. Hellerstein, J. F. Naughton, and J. F. Naughton, "Generalized search trees for database systems," in *Proceedings of the 21st VLDB Conference*. Zurich, Switzerland: Morgan Kaufmann Publishers Inc., 1995.

 ISBN: 978-1-61208-162-5