An Efficient Bi-Dimensional Indexing Scheme for Three-Dimensional Trajectories

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

Marco Leone, Alessia Saggese,
Mario Vento
DIEII, University of Salerno
Fisciano (SA), Italy
{*mleone,asaggese,mvento*}*@unisa.it*

***Abstract*—The proliferation of devices able to monitor their position is favoring the accumulation of large amount of geographically referenced data, that can be profitably used in a lot of applications, ranging from traffic control and management to location-aware services. The strong interest in these applications has entailed a significant research effort in the last years, both toward the modeling of spatio-temporal databases and toward indexing strategies to efficiently process spatio-temporal queries. Recently, we presented an indexing scheme (based on a redundant storing strategy) able to index three-dimensional trajectories using widely available bidimensional indexes. In this paper we propose a method that, while avoids redundant storing of data, still uses well established bi-dimensional indexes. With respect to the previous work, the retrieving performance is improved by taking advantage both of a more efficient representation and of a trajectory segmentation stage, as experimental results show.**

Keywords-indexing; moving objects databases; spatial queries.

I. INTRODUCTION

The increasing number of mobile devices able to report their position in real-time with high accuracy has implied the collection of large amount of data, which can be profitably used in many realms, ranging from traffic control and management to location-aware services [1][2]. Moreover, the need for security in many public environments has also contributed to an exponential proliferation in the number of available cameras and, starting from the video streams acquired from these peripherals, objects' positions can be extracted by using available video analytic algorithms [3]. In this way, databases for the analysis and the validation of models related to different typologies of objects' movements and behaviors (pedestrians, cars, pets and so on) have gained great interest.

In order to store and efficiently retrieve the information extracted from this large amount of acquired data, in the last years a significant research effort has been made, both towards the modeling of spatio-temporal Moving Object Databases (MODs) and towards indexing strategies aimed at efficiently process spatio-temporal queries.

According to the widely adopted line segment model, it is straightforward to represent the movement of each object as a sequence of line segments, each represented by two sample positions at consecutive time instants. The trajectory associated to the object's motion is thus represented by a polyline in a three-dimensional space, the first two dimensions referring to the space and the third one to the time. An example is shown in Figure 1.

When handling with MODs, we typically aim at extracting, from the entire collection of stored data, only those trajectories possessing a given property: information retrieval is therefore achieved through processing the query submitted by the user.

Queries that are worth to be considered in spatio-temporal databases can be subdivided at least into three orthogonal categories. First, from a temporal perspective, *"find all the vehicles that will be in a given area in the next ten minutes"* is an instance of the so-called *future query*; in order to solve this query, models able to predict the future position of a moving object are needed. On the other hand, *past queries* handle with the historical positions of moving objects. Finally, *now queries* ask for the position of objects at the current time instant t^{now}; these queries can be considered as a special case of future queries if the last recorded position is lower than t^{now}; furthermore, they represent a special case (at least from the indexing strategy point of view) of now queries if the current position of objects has been recorded.

Another commonly accepted query taxonomy [4][5] is based on the following consideration: a trajectory is a very complex data structure, so implying that the time needed to extract it from the database strongly influences the performance of a generic retrieval system. For this reason, it is important to distinguish, for example, if we are only interested in the identifiers of the objects passing through a given area in a given time interval or if we are interested in the whole trajectory of these objects. The former query is commonly defined as *coordinate-based query*; a typical example is: *'find the number of pedestrians in Saint Peter's square in Rome between 9-12 am yesterday'*; the latter is commonly defined as *trajectory-based* query and, in turn, contains two different categories: *topological queries* and *navigational queries*. Topological queries make use of information about the scene for the extraction of an object's trajectory (*'when did vehicle X leave Plebiscito's square in Naples most recently?'*). Navigational queries need derived

information to answer questions like *'what is the current speed of vehicle X?'*; in this case, the needed information (like speed, heading, travel distance, etc.) is not typically stored directly and, therefore, a computational overhead is paid.

Furthermore, from a temporal point of view, a simple but useful distinction is made between time slice and time interval queries; a *time slice query* asks about a fixed time point while a *time interval query* considers a temporal interval. For the sake of clarity, a *time slice query*, in its coordinate-based form, can be exemplified as *'find all the objects that are in a given area at a given time instant* t_i*'* while its trajectory-based form is *'find the trajectories of each object that is in a given area at a given time instant* t_i*'*. When $t_i < t_{now}$, we are facing a *past query*; $t_i > t^{now}$ characterizes *future queries*, while $t_i = t^{now}$ is the case of *now queries*. A *time interval query*, in its coordinate-based form, can be exemplified as *'find all the objects that pass through a given area in the time interval* $[t_1, t_2]$*'*, while a trajectory-based example is *'find all the trajectories of objects that traverse a given area in the time interval* $[t_1, t_2]$*'*. Assuming $t_1 < t_2$, if $t_2 < t^{now}$ we are dealing with a *past query*, while $t_1 > t^{now}$ characterizes *future queries*. The case $t_1 < t^{now} < t_2$ can be easily assumed as composed by a past time interval query ($t \in [t_1, t^{now})$, a now time slice query ($t = t^{now}$) and a future time interval query ($t \in (t^{now}, t_2]$).

Many other interesting queries are reported in [6], [7], [8], [9] and in [10]. The large amount of queries that have been proposed, especially in the last years, reinforces the evidence that research is still ongoing in this field and, at our knowledge, efficient solutions are still being investigated.

In this context, we are interested in efficiently storing and querying moving objects' trajectories extracted from video cameras. Although efficient bidimensional indexing methods are usually available, several problems arise when data to be handled are three- or even four-dimensional, as it happens for the trajectory-based systems. To solve these problems, we recently proposed a method [1] able to redundantly project and analyze a collection of trajectories on bi-dimensional planes by using off-the-shelf solutions.

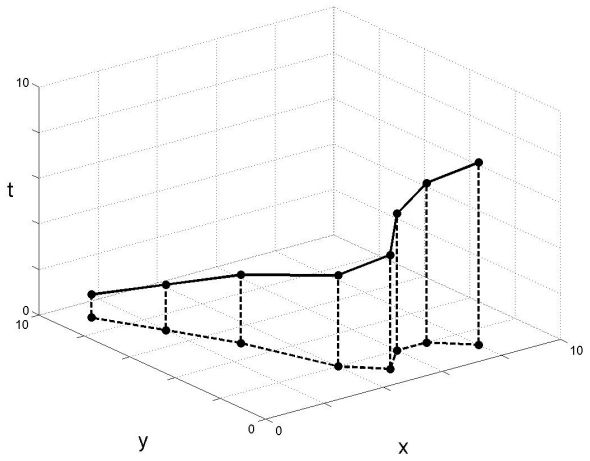


Figure 1. A spatio-temporal trajectory; *x* and *y* dimensions refer to position while the third dimension (*t*) refers to time.

Starting from our previous work, in this paper we propose an improved version of the system able to answer past (trajectory-based) time interval queries on a MOD; even still using bidimensional indexes, the proposed solution avoids the redundancy in the stored data and improves the whole performance, also thanks to a segmentation algorithm aimed at optimizing the use of the adopted indexes.

The paper is organized as follows: after a discussion about some of the papers related to both bi-dimensional and three-dimensional data indexing methods (Section II), in Section III we describe our previous proposal in [1] and its improvement, obtained by using a new indexing strategy that avoids redundancy in the data to be stored; in Section III we also introduce and describe the adopted segmentation algorithm. Experimental results are the concern of Section IV, while Section V concludes the paper outlining future directions.

II. Related Work

Indexing moving objects databases has been an active research area in the recent past and several solutions have been proposed. [6] and [11] survey many accessing strategies, proposed in the last two decades, which are able to index the past and the current position, as well as methods supporting queries related to the future.

According to [11] and [12], one of the most influential accessing methods in the area of spatial data management was proposed by Guttman. He suggested, in his pioneering paper [13], a structure named R-tree able to efficiently index bidimensional rectangular objects in VLSI (Very Large Scale Integration) design applications. The conceptual simplicity of an R-tree and its resemblance to widely adopted standard B-trees, allowed the developers to easily incorporate such a solution in spatial enabled DBMS [12] in order to support spatial query optimization and processing. R-trees hierarchically organize the geometric objects by representing them through *Minimum Bounding Rectangles* (MBRs, [14]), which are an expression of the object's maximum extents in its coordinate system; each internal node corresponds to the MBR that bounds its children while, as usual, a leaf contains pointers to the objects (see Figure 2). The insertion of a new object takes place by choosing, at each level, the node that involves the smallest expansion; when a split of the selected leaf node is needed, Guttman proposed three algorithms with different complexity to handle such a split, aiming at the minimization of the sum of the areas of resulting nodes. It is worth noting that, since an MBR can be included in many

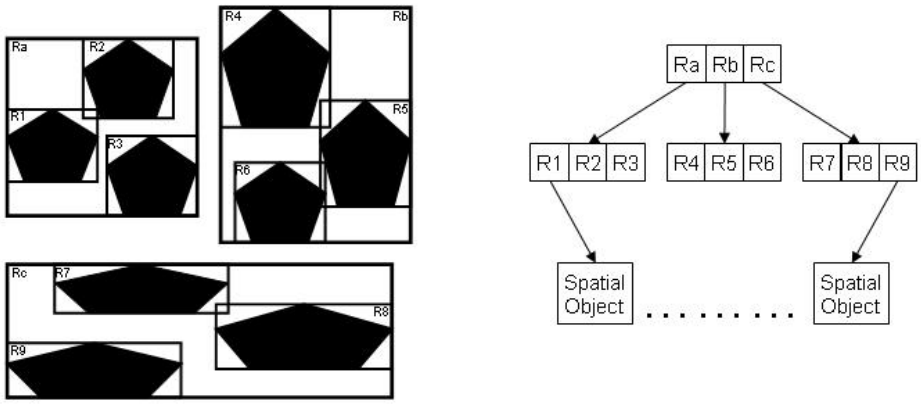


Figure 2. An R-tree example (adapted from [12]).

nodes (see R6 in Figure 2), a spatial search may include the visit of many nodes.

Starting from the original R-Tree structure, several improved versions have been proposed; when we move from spatial to spatio-temporal data, for instance, the temporal coordinate can be considered as an extra dimension and data can be indexed using three-dimensional R-trees [15]. Such an approach does not discriminate between spatial and temporal dimensions and is well suited for indexing the past, i.e., when only closed trajectories are considered.

STR-trees [4] extend R-tree with a different insert/split algorithm, while the characteristics of spatio-temporal data are captured by two access methods (STR-tree and TB-tree).

When objects' movements are constrained, for example on a network of connected road segments, a bidimensional R-tree can be used to index the static network's segments. In this case, each leaf contains a segment and a pointer to a monodimensional R-Tree that indexes the time intervals of objects' movements, as for FNR-Tree [16]. MON-tree [17] extends the FNR-tree by modeling the constrained network as a set of junctions and routes; a bidimensional R-tree is used to index polylines' bounding boxes while, for each polyline, another bidimensional R-tree indexes the time dimension of the objects within the polyline. PARINET [18] has been designed for historical data in constrained networks and models the network as a graph; trajectories are partitioned according to the graph partitioning theory. This method has been extended to handle continuous indexing of moving objects [19].

When dealing with real applications for indexing and querying large repositories of trajectories, the size of MBRs can be reduced by segmenting each trajectory and then indexing each sub-trajectory by using R-Trees; such an approach is described, for example, in [20], where a dynamic programming algorithm is presented for the minimization of the I/O for an average size query. SETI [21] 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 [22] co-locates on a disk block (or in a collection of adjacent blocks) trajectory' segments by using an adaptive multi-level grid; thanks to this method, it is possible to answer a query by only reading a few blocks.

All the above approaches, even presenting efficient solutions from different perspectives, are typically not supported in the available commercial products that make use of very efficient spatial indexes that, unfortunately, are typically restricted to the bi-dimensional case. For instance, PostGIS [23], a well known extension of PostgreSQL DBMS [24] for storing spatial data, even supporting three (and four)-dimensional data, does not support three-dimensional intersection and indexing operations. As a consequence, there is a strong interest in those methods which, even using off-the-shelf solutions, allow to solve the problem in the multi-dimensional space.

For this reason, in this work we propose a method able to index large sets of trajectories extracted from video cameras by means of off-the-shelf solutions; in our approach, we consider that objects are freely moving in our scene, without any kind of constraint. The main novelty lies in the fact that, differently from previous solutions, we do not suggest any new indexing three-dimensional structures. As a matter of fact, we propose a way to efficiently use available bidimensional solutions in order to solve the problem that spatial indexes for three-dimensional data are not widely available.

III. The Proposed Method

According to the line segment model, a trajectory T^k can be represented as a sequence of spatio-temporal points:

$$T^k =< P_1^k, P_2^k, ..., P_n^k >$$

with:

$$P_i^k = (x_i^k, y_i^k, t_i^k) \quad \forall i \in [1, n].$$

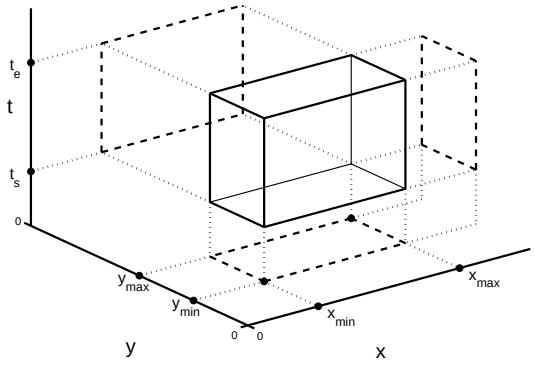


Figure 3. A query box representing a *TIQ*.

Each pair (x_i^k, y_i^k) refers to the spatial location of an object at the time instant t_i^k. As already mentioned, the trajectory is approximated by a polyline, each segment being the linear interpolant between two consecutive points (Figure 1).

A *Time Interval Query* (*TIQ*) aims at detecting all those trajectories passing through a given spatial area A in a given time interval $[t_s, t_e]$; assuming that the area A is rectangular, the latter is fully identified by two points in the xy plane $P_m^{xy} = (x_{min}, y_{min})$ and $P_M^{xy} = (x_{max}, y_{max})$. Each *TIQ* can be thus associated to a query box B:

$$B = \{(x_{min}, y_{min}, t_s), (x_{max}, y_{max}, t_e)\},$$

Differently speaking, the temporal dimension extends the rectangular area A in the 3D space (see Figure 3).

From a geometrical point of view, in order to solve a *TIQ* we need to find all those trajectories intersecting the query box B. A simple algorithm for retrieving the trajectories satisfying such a query is based on processing, for each trajectory, all its segments, starting from the first one: as soon as the intersection occurs, it can be concluded that the trajectory intersects the query box. In order to determine if a trajectory segment lies inside or outside a query box, a clipping algorithm can be used.

We propose to use the 2D Cohen-Sutherland Line Clipping Algorithm [25] (briefly summarized in Figure 4). According to it, the geometric plane is subdivided into nine areas by extending the edges of the query rectangle: if at least one of the segment endpoints lies inside the query box, the intersection is trivially verified (see segment AB in Figure 4.a); if, on the contrary, both the endpoints lie outside the query box, we check the position of the endpoints with respect to the query area: in some cases the intersection can be still trivially verified, as for CD and EF respectively in Figure 4.b, otherwise the segment is split at its intersection points and each obtained sub-segment is in turn inspected (as in the case of the segment GH and IL in Figure 4.c).

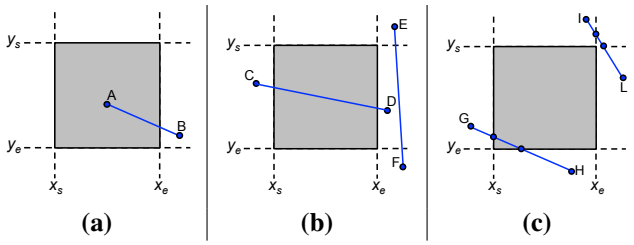


Figure 4. The Cohen-Sutherland Algorithm [25].

This clipping algorithm can be easily extended for dealing with 3D trajectories by considering 27 spatial regions, rather than 9.

Unfortunately, despite its simplicity, the use of a clipping algorithm is not suited for handling with large datasets, so demanding for more efficient approaches. In fact, the clipping algorithm has to process, for each trajectory, all its segments, starting from the first one until the intersection occurs. The worst case arises when a trajectory does not intersect at all the query box; in this case, in fact, all its segments must be processed, making this approach infeasible for large amount of data.

When the number of trajectories increases, more efficient approaches are thus mandatory; for example, suitable indexing strategies would be necessary to reduce the number of trajectories to be clipped. Although many spatial databases today available, both open source and commercial ones, provide very efficient spatial indexing techniques, these indexes schemes are unfortunately typically restricted to deal with 2D data; for this reason, it is worth trying to represent the actual 3D problem in terms of (one or more) 2D sub-problems, so as to fully exploit the efficient available bidimensional indexes.

A. The Solution Proposed in [1]

In [1], given a trajectory T^k and a query box B, we proposed a method according to which both T^k and B were first projected on the three coordinate planes; let T_{kz}^k and B_{kz} be the projections of T^k and of B on the kz plane. We then observed that, if a trajectory intersects the 3D query box, then each trajectory projection must also intersect the correspondent query box projection:

$$T^k \cap B \neq \emptyset \Rightarrow \left\{ \begin{array}{c} T_{xy}^k \cap B_{xy} \neq \emptyset \\ T_{xt}^k \cap B_{xt} \neq \emptyset \\ T_{yt}^k \cap B_{yt} \neq \emptyset \end{array} \right\} \quad (1)$$

Equation 1 represents a necessary but not sufficient condition, as the opposite is clearly not true. In fact, if all trajectory's projections intersect the correspondent box projection on the considered spaces, they do not necessarily intersect the 3D query box too. To better explain this concept, Figure 5.a shows, in the 3D space, a trajectory that does not intersect a given query box: it can be noticed that all the

trajectory projections intersect the correspondent query box projections (Figure 5.b-d).

Thus, as a matter of fact, if all projections of T^k intersect the correspondent box projections, we consider T^k as a candidate to be clipped in the three-dimensional space; the guess is that there will be not too many *false positives*.

According to the above considerations, in [1], for each three-dimensional trajectory T^k, we proposed to store three bi-dimensional trajectories obtained by projecting T^k on the xy plane (T^k_{xy}), on the xt plane (T^k_{xt}) and on the yt plane (T^k_{yt}).

Given a box B representing the time interval query to be solved, we similarly considered B_{xy}, B_{xt} and B_{yt}.

With this strategy, by using one of the available bi-dimensional indexes, it is possible to find on each plane the following three trajectory sets (Θ_1, Θ_2, Θ_3) in a very simple and efficient manner:

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

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

$$\Theta_{yt} = \{T_{yt} : MBR(T_{yt}) \cap B_{yt} \neq \emptyset\} \quad (4)$$

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

$$\Theta = \{T : T_{xy} \in \Theta_{xy} \wedge T_{xt} \in \Theta_{xt} \wedge \Theta_{yt} \in T_{yt}\} \quad (5)$$

This strategy, while taking advantage of widely available efficient bidimensional indexes, still presents two weak points. First, for a n points trajectory, we need to redundantly store $6{\cdot}n$ values ($2{\cdot}n$ for each of the three coordinate planes). Another subtle crucial point is that the use of bidimensional indexes is not optimized: as a matter of fact, the MBR of each projected trajectory can easily span a great percentage of the whole area.

In the following two subsections, the above problems will be separately handled and solutions for them will be presented.

B. Improving the Method by Removing Redundancies

It is possible to observe that, for a given trajectory T^k, rather than storing the three different trajectory projections in each coordinate plane, we can store T^k as the original sequence of points in the 3D space, and separately maintain three different bidimensional MBRs: $MBR_{xy}(T^k)$, $MBR_{xt}(T^k)$ and $MBR_{yt}(T^k)$. $MBR_{xy}(T^k)$ (respectively $MBR_{xt}(T^k)$ and $MBR_{yt}(T^k)$) is obtained by projecting on the xy (respectively xt and yt) plane the three-dimensional MBR of T^k.

It is worth noting that the redundancy introduced by the three MBR projections is not dependent on the number of points in the trajectory and, therefore, has only a marginal impact on the spatial complexity, since it only requires the storage of six pairs of points.

Assuming such a scheme, on each 2D plane we find the trajectories intersecting the corresponding 2D query box in a very efficient manner by using one of the available 2D indexes. Let Γ_{xy}, Γ_{xt} and Γ_{yt} be the resulting sets of trajectories defined as:

$$\Gamma_{xy} = \{T : MBR_{xy}(T) \cap B_{xy} \neq \emptyset\} \quad (6)$$

$$\Gamma_{xt} = \{T : MBR_{xt}(T) \cap B_{xt} \neq \emptyset\} \quad (7)$$

$$\Gamma_{yt} = \{T : MBR_{yt}(T) \cap B_{yt} \neq \emptyset\}, \quad (8)$$

where, as usual, B_{xy}, B_{xt} and B_{yt} are the projections of the 3D query box B. The set Θ of the trajectories candidate to be clipped in the 3D space is therefore now defined as:

$$\Theta = \Gamma_{xy} \cap \Gamma_{xt} \cap \Gamma_{yt} \quad (9)$$

Figure 6 resumes the method: given a set of trajectories and a query box (Figure 6.a), we discard the green trajectory since its $MBR_{xy}(T)$ and $MBR_{yt}(T)$ do not intersect the corresponding projection of the black query box (Figure 6.b and Figure 6.c). The candidate set Θ is thus composed by the other two trajectories, the red and the blue one, which are finally clipped, obtaining the desired output represented by the blue trajectory (Figure 6.e).

C. Optimizing the Selectivity of the 2D Indexes

It should be clear at this point that the entire system performance will strongly depend on the indexing phase and, as a consequence, on the capability to reduce the number of trajectories to be clipped in the three-dimensional space. At a more detailed analysis, the *selectivity* of the indexes in each plane is related to the area of the corresponding MBR which, in turn, only depends on the trajectory geometry, so being (apparently) fixed. This is the reason why we decided to introduce a segmentation stage, aimed at increasing the selectivity of the indexes.

Segmentation algorithms aim at subdividing each trajectory into consecutive smaller units, which we will refer to as trajectory units. We are interested in a segmentation algorithm able to exploit the characteristics of the available bi-dimensional indexes; this can be accomplished by decreasing the area of the projected MBR of each trajectory unit.

The proposed algorithm works recursively: initially (that is at iteration 0) it assumes that the trajectory T^k is composed by a single unit ${}^0U^k_1$, that is split into a set of m consecutive smaller units $\{{}^1U^k_1, \ldots, {}^1U^k_m\}$; each of the ${}^1U^k_i$ is in turn inspected and, if the stop criteria are not satisfied, it is further split.

Let us analyze how a generic unit ${}^{(i-1)}U = \{P_1, \ldots, P_m\}$ is split into $\{{}^iU_1, \ldots, {}^iU_n\}$; we first choose a *split-dimension* and a *split-value*. Assume, as an example and without loss of generality, that x has been chosen as the *split-dimension* and let x^* be the *split-value*. In addition, assume that $x_1 < x^*$. According to these hypotheses, ${}^iU_1$

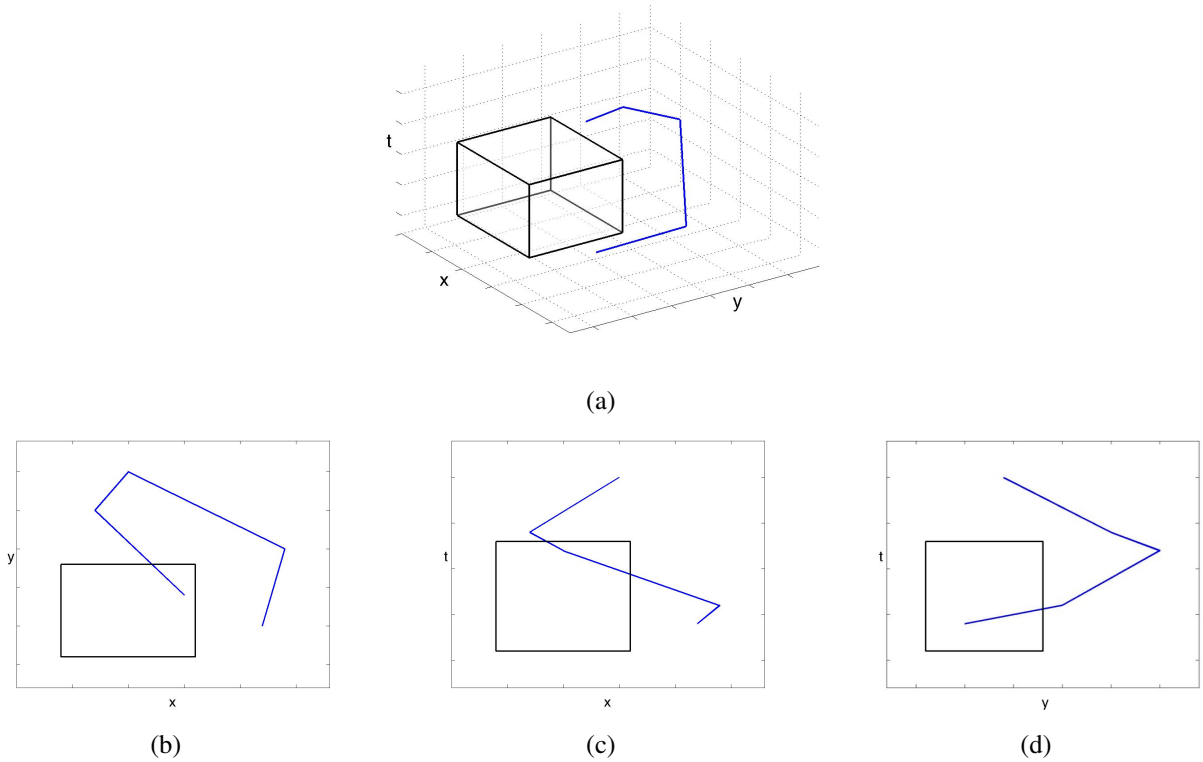


Figure 5. An example of 3D trajectory (a) and its projections on the different coordinate planes xy (b), xt (c) and yt (d). Although the trajectory does not intersect the query box, its projections do it.

is the set of the consecutive points lying on the left of the *split-value*:

$$^{i}U_1 = \{P_1, \ldots, P_k\} \tag{10}$$

where P_k is the first point such that $x_k \geq x^*$. Then, the second unit will be formed by the sequence of consecutive points lying on the right of the *split-value*:

$$^{i}U_2 = \{P_{k+1}, \ldots, P_l\} \tag{11}$$

where P_l is the first point such that $x_k \leq x^*$. The inspection of $^{(i-1)}U$ ends when the last point P_m is reached.

According to the above considerations, the criteria for the choice of the two parameters, *split-dimension* and *split-value*, play a crucial role. Since we aim at optimizing the indexing strategy, the proposed segmentation algorithm is based on the occupancy percentage on each 2D coordinate plane.

First we calculate the coordinate plane corresponding to the maximum among the three occupancy percentage values O_{xy}, O_{xt} and O_{yt} of the trajectory unit MBRs, with respect to the correspondent global volume of interest V:

$$O_{xy} = \frac{MBR^{xy}(U)}{V^{xy}} \tag{12}$$

$$O_{xt} = \frac{MBR^{xt}(U)}{V^{xt}} \tag{13}$$

$$O_{yt} = \frac{MBR^{yt}(U)}{V^{yt}} \tag{14}$$

Without loss of generality, suppose that the maximum occupancy percentage value is O_{xy} and, consequently, the corresponding plane is xy; let $width$ and $height$ be the two dimensions of $MBR_{xy}(U)$, respectively along the coordinates x and y; the *split-dimension* sd is defined as:

$$sd = \begin{cases} x & \text{if width} > \text{height} \\ y & \text{otherwise} \end{cases}$$

Given the *split-dimension* sd we choose, as the *split-value* sd^*, the MBR average point on the coordinate sd.

Figure 7 sketches the execution of the first iteration of our algorithm on the trajectory T (assumed to be composed at this iteration by a single unit U).

In Figure 7.a we are assuming that our volume of interest is:

$$0 \leq x \leq 150; 0 \leq y \leq 50; 0 \leq t \leq 30 \tag{15}$$

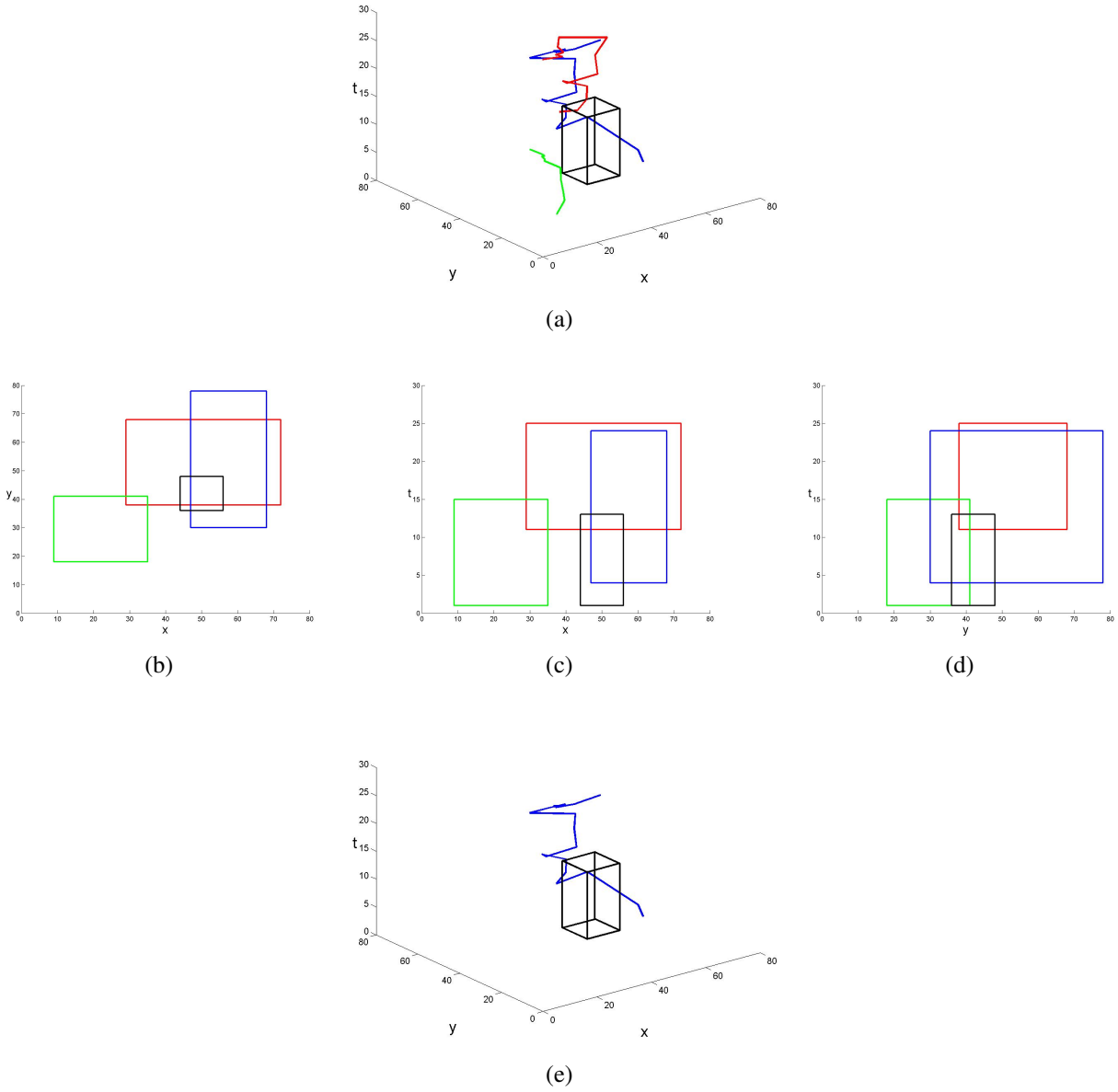


Figure 6. An overview of the method. (a) a query box and three trajectories; (b), (c), (d): the projections of trajectories' MBRs on the coordinate planes; (e) the final result of our method, after the application of the clipping algorithm on the blu and red trajectories.

We first consider $MBR_{xy}(U)$, $MBR_{xt}(U)$ and $MBR_{yt}(U)$ (Figure 7.b) obtaining that:

$$O_{xy} > O_{xt} > O_{yt}. \tag{16}$$

According to the above inequalities, we choose to operate on the xy plane. As the values of the dimensions are:

$$\text{width} = x_{max} - x_{min} = 135 \tag{17}$$

$$\text{height} = y_{max} - y_{min} = 40, \tag{18}$$

assuming that x has been chosen as the split-dimension, the split-value will be easily obtained as follows:

$$x^* = \frac{x_{max} + x_{min}}{2} = 72.5 \tag{19}$$

$$\tag{20}$$

Now we are in the position of segmenting the unit; Figure 7.c shows the current iteration, that is the segmentation of the unit while, in Figure 7.d, are shown, with different colors, the obtained 4 units with the corresponding MBRs. Last,

Figure 7.e shows the projections of the obtained MBRs on each coordinate plane.

The algorithm ends when all the trajectory units cannot be further subdivided, since at least one of the stop conditions has been reached for each unit; in particular, we employ two stop criteria . First, we choose not to segment trajectory units whose MBR areas are smaller than a fixed percentage of the entire scenario (PA^{min}); furthermore, we do not segment a unit with less than PS^{min} points.

Finally, a further refinement is needed in our algorithm for handling with the formation of the last unit; in fact, when the generic unit U^i is splitted, it can happen that the last unit is only composed by a few points. In this case, a suited strategy is introduced to specifically handle with this issue. Figure 8.a clarifies this concept: the last unit (the black one) is composed by three points, but LU^{min} is set to four. For this reason, this unit will be merged with the previous one (the green one, see Figure 8.b).

D. Summarizing the method

In this section we will summarize the proposed method. Each trajectory T^k is segmented (when it is loaded) and stored as a sequence of trajectory units $\{U_1^k, \ldots, U_l^k\}$; using one of the available bidimensional indexes, we select:

$$\Gamma_{xy}^U = \{U : MBR_{xy}(U) \cap B_{xy} \neq \emptyset\} \quad (21)$$

$$\Gamma_{xt}^U = \{U : MBR_{xt}(U) \cap B_{xt} \neq \emptyset\} \quad (22)$$

$$\Gamma_{yt}^U = \{U : MBR_{xy}(U) \cap B_{yt} \neq \emptyset\}, \quad (23)$$

where, as usual, B_{xy}, B_{xt} and B_{yt} are the projections of the 3D query box B.

The set Γ^U of units candidate to be clipped in the 3D space is therefore defined as:

$$\Gamma^U = \Gamma_{xy}^U \cap \Gamma_{xt}^U \cap \Gamma_{yt}^U. \quad (24)$$

By analyzing Γ^U we build, as follows, the set Θ:

$\Theta = \{\emptyset\}$
$\forall\ U^k \in \Gamma^U\ \{$
  if $k \notin \Theta\ \{$
    if U^k intersects B in the three-dimensional space $\{$
      $\Gamma = \Gamma \bigcup k$
    $\}$
  $\}$
$\}$.

The set Θ represents the result of a TIQ, if we are interested in the coordinate-based form, while the entire trajectories have to be extracted if we are interested in trajectory-based TIQs.

IV. Experimental Results

In order to characterize the efficiency of the proposed method, several trajectory-based *TIQs* were performed. We conducted our experiments on a PC equipped with an Intel quad core CPU running at 2.66 GHz, using the 32 bit version of PostgreSQL 9.1 server and the 1.5.3 version of PostGIS. Data have been indexed using the standard bidimensional R-tree over GiST (Generalized Search Trees) indexes; as the specialized literature confirms, this choice guarantees higher performance in case of spatial queries with respect to the PostGIS implementation of R-trees.

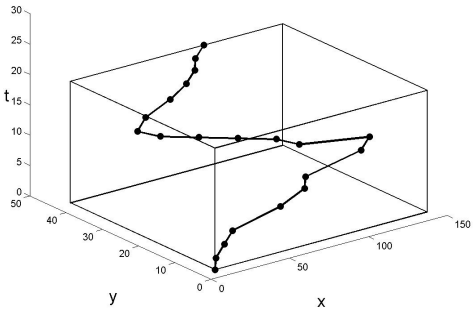
We represent each trajectory unit as a tuple:

$$(\underline{ID}, \underline{UID}, U, MBR_{xy}, MBR_{xt}, MBR_{yt}) \quad (25)$$

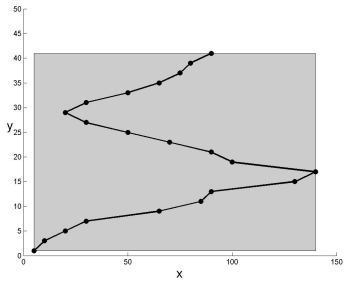
where ID is the moving objects identifier, UID identifies the trajectory unit, and U is the 3D trajectory unit, represented as a sequence of segments (a PostGIS 3D multi-line). Finally MBR_{xy}, MBR_{xt} and MBR_{yt} are the three unit's MBRs in each coordinate plane, xy, xt and yt respectively, represented as PostGis *BOX* geometries. Once data have been indexed, PostGIS provides a very efficient function to perform intersections between boxes and MBRs in a 2D space.

The experimental results have been obtained by testing the system performance on synthetic data, which have been generated as follows. Let W and H be the width and the height of our scene and S the temporal interval. Each trajectory T^i starting point is randomly chosen in our scene at a random time instant t_1^i; the trajectory length L^i is assumed to follow a Gaussian distribution, while the initial directions along the x axis and the y axis, respectively d_x^i and d_y^i, are randomly chosen. At each time step t, we first generate the new direction, assuming that both d_x^i and d_y^i can vary with probability PI_x and PI_y respectively; subsequently, we choose the velocity along x and y at random. The velocity is expressed in pixels/seconds and is assumed to be greater than 0 and less than two fixed maximum, V_x^{max} and V_y^{max}. Therefore, the new position of the object can be easily derived; if it does not belong to our scene, new values for d_x and/or d_y are generated. We refer to the scene populated with trajectories as the *Scenario*. Table I reports the free parameters and the values for the creation of the 30 different scenarios used in our experiments as well as the parameters used by the segmentation algorithm. Note that the worst case, corresponding to the maximum values of T and L, results in 10^4 trajectories with 10^4 points, for a total of 10^8 points to store and process; these values are over and above if compared with many real world datasets.

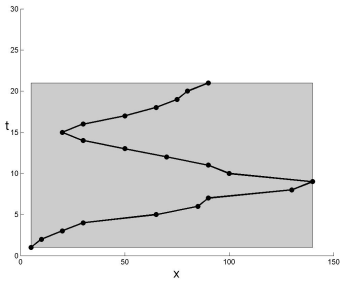
For the evaluation of the efficiency of the proposed segmentation algorithm, we generated and segmented 6000 trajectories with $L \in \{1000, 2000, 3000, 4000, 5000, 10000\}$; for each trajectory T^i we measured the number of obtained segments (N_{seg}^i) and the time needed to segment the trajectory (T_{seg}^i). Last, the obtained N_{seg}^i and T_{seg}^i are averaged over L, so obtaining $\overline{N_{seg}}(L)$ and $\overline{T_{seg}}(L)$. Figure 9 shows on the left the averaged number of segments ($\overline{N_{seg}}$) as L changes; not surprisingly we have, with very



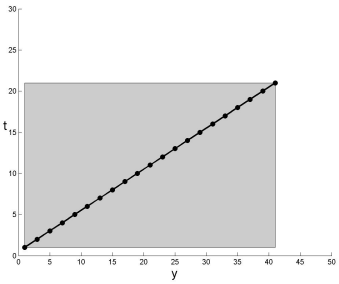
(a)



$O_{xy} = 72\%$

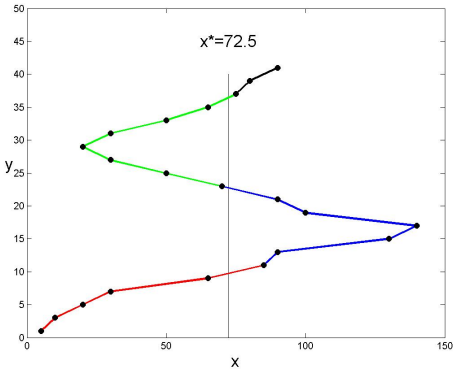


$O_{xt} = 69\%$

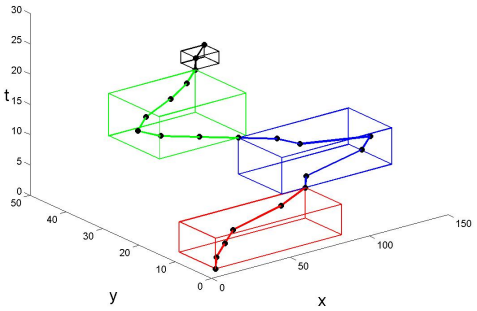


$O_{yt} = 53.4\%$

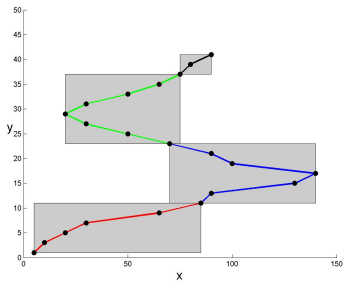
(b)

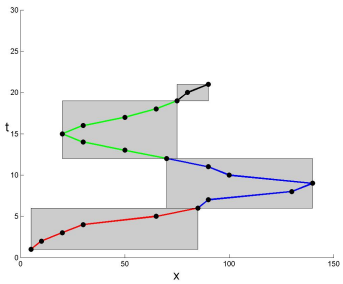


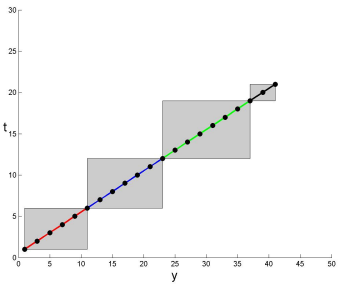
(c)



(d)

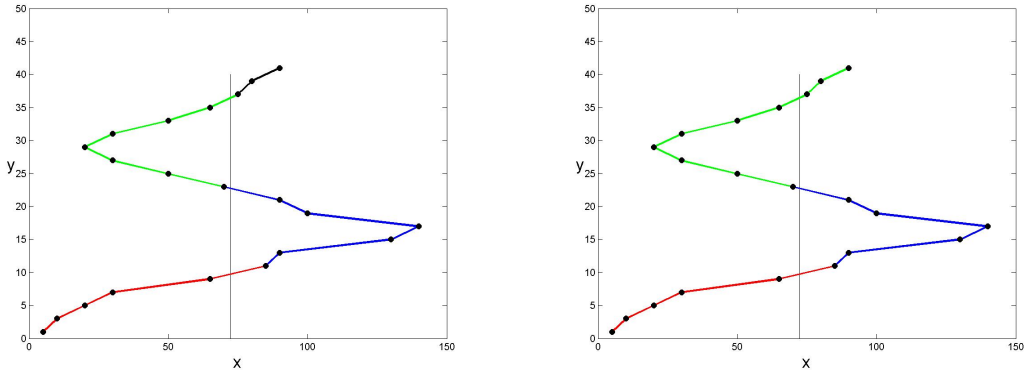






(e)

Figure 7. An overview of the segmentation algorithm. See text for details.



(a) (b)

Figure 8. The effect of LU_{min} on a segmented unit.

Table I
THE PARAMETERS USED IN OUR EXPERIMENTS.

Scene width (pixels)	10^4
Scene height (pixels)	10^4
Time interval length (secs)	10^5
Number of trajectories (T)	$\{1, 2, 3, 5, 10\} * 10^3$
Points in each trajectory (L)	$\{1, 2, 3, 5, 10\} * 10^3$
PI_x	5%
PI_y	5%
V_x^{max}	10 pixels/secs
V_y^{max}	10 pixels/secs
PA^{min}	1%
PS^{min}	100
LU^{min}	10

good approximation, that the number of segments linearly increases with L. On the right of Figure 9 is shown, in milliseconds, the averaged time $\overline{T_{seg}}$ needed to segment a trajectory with L points; with good approximation, $\overline{T_{seg}}$ quadratically increases with L.

The time needed to process a generic *TIQ* query (QT) is a function of at least 4 parameters, namely the number of trajectories T, the average trajectories' length L, the query cube dimension D_c, expressed as percentage of the entire scenario, and the position of the query box P_c:

$$QT = f(T, L, D_c, P_c). \quad (26)$$

Among the above parameters, P_c strongly influences the time needed to extract the trajectories as these are not uniformly distributed, especially in real world scenarios. In order to avoid the dependency on the query cube position, we decided to repeat the query a number of times inversely proportional to the query cube dimension, positioning the query cube in different positions, as shown in row N of Table II; finally, results are averaged to obtain:

$$\overline{QT} = f(T, L, D_c). \quad (27)$$

For the description of the experimental results we define three different set of experiments, obtained by fixing two

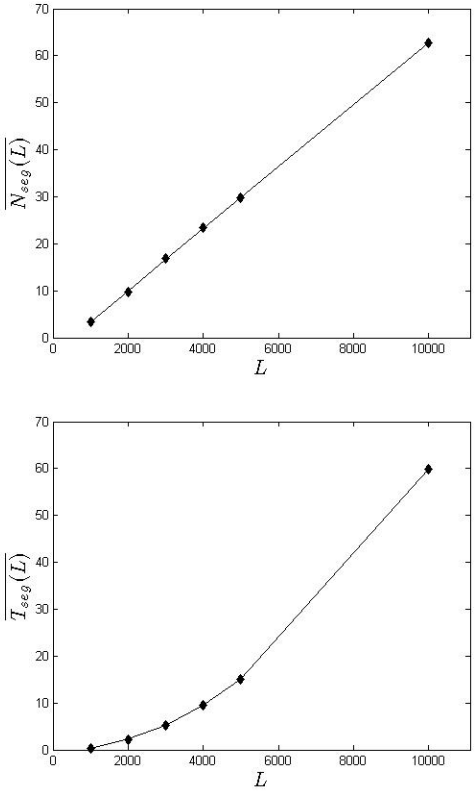


Figure 9. The performance of the segmentation algorithm.

Table II
NUMBER N OF TIMES EACH QUERY IS REPEATED AS D_c VARIES.

D_c	1%	5%	10%	20%	30%	50%
N	200	40	20	10	7	4

of the three parameters T, L and D_c and showing the variation of $\overline{QT}$ with respect to the third (free) parameter; an example is shown in Figure 10, which expresses the values

of $\overline{QT}$ with D_c and L fixed and T variable. Each diamond refers to the real value in seconds of $\overline{QT}$ for a given value of T; Figure 10 shows the curve which best interpolates the diamonds: in this case the approximation is linear, so obtaining a line. It is worth pointing out that, for the sake of readability, the figures for the experimental results will be expressed in a semi-log scale as, even not permitting to display part of the curves interpolating small values, it provides a greater comprehension of the system behavior for large values of the parameters.

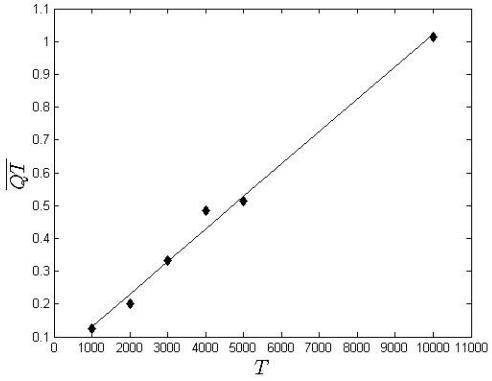


Figure 10. $\overline{QT}$ (in seconds) as the number of trajectories increases with $D_c = 5\%$ and $L = 5000$.

In Figure 11, $\overline{QT}$ relates to the variable number of trajectories T, for various values of D_c; the number of curves corresponds to the different fixed values of $L = \{1, 2, 3, 5, 10\} * 10^3$. The relationship between $\overline{QT}$ and T has been analyzed by polynomially approximating $QT(T)$: note that $\overline{QT}$ linearly increases with T, with a very small factor of approximation.

Diamond points in Figure 12 express $\overline{QT}$ in relation to the query box dimensions D_c and for $T = 3.000$ and $T = 10.000$; the number of curves corresponds to the different fixed values of $L = \{1, 2, 3, 5, 10\} * 10^3$. In this case we obtain that $\overline{QT}$ quadratically depends on D_c.

In Figure 13, the diamonds express $\overline{QT}$ as a function of L, for $D_c \in \{1\%, 5\%, 20\%, 30\%\}$, while the number of curves corresponds to the different fixed values of $T = \{1, 2, 3, 5, 10\} * 10^3$. Again we obtain that $\overline{QT}$ quadratically depends on L.

Finally, Figure 14 highlights the enhancement of the proposed indexing scheme as compared with the one presented in [1], for $D_c \in \{1\%, 20\%\}$. In particular, the diamonds refer to the new method, while the circles refers to the previous one. Note that, thanks to a novel indexing strategy and segmentation algorithm, the system performance significantly improved while the redundancy in stored data has been significantly removed.

V. Conclusion

In this paper we proposed an enhanced version of the retrieval system proposed in [1], aimed to index large amount of 3D trajectory data by using widely available 2D indexes. With respect to the previous method, two main improvements have been achieved: the former is the removal of the redundancy, obtained thanks to a novel indexing scheme; the latter is the optimization of the selectivity of the indexes, obtained by introducing a segmentation algorithm.

The experimental results, performed on synthetic data, show that the proposed solution is able to fully exploit the retrieving capabilities based on well established 2D indexes. In fact, the system performance have been significantly improved, if compared with the results presented in [1].

Further improvements in the performance will be achieved by applying the clipping algorithm in parallel to each candidate trajectory. This step can be easily implemented using multi-threading, in order to take advantage of multi-core and multi-processors systems. Strategies aiming at compressing data to be stored and retrieved are also being considered. Finally, we are extending our system in order to answer different query typologies.

Acknowledgment

This research has been partially supported by A.I.Tech s.r.l. (a spin-off company of the University of Salerno, www.aitech-solutions.eu)) and by the FLAGSHIP InterOmics project (PB.P05, funded and supported by the Italian MIUR and CNR organizations).

References

[1] A. d'Acierno, A. Saggese, and M. Vento, "A redundant bi-dimensional indexing scheme for three-dimensional trajectories," in *Proceedings of the First Conference on Advances in Information Mining and Management (IMMM 2011)*, 2011.

[2] A. d'Acierno, M. Leone, A. Saggese, and M. Vento, "A system for storing and retrieving huge amount of trajectory data, allowing spatio-temporal dynamic queries," in *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, sept. 2012, pp. 989 –994.

[3] R. Di Lascio, P. Foggia, A. Saggese, and M. Vento, "Tracking interacting objects in complex situations by using contextual reasoning," in *Computer Vision Theory and Applications (VISAPP), 2012 International Conference on*, 2012.

[4] 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.

[5] D. Lim, D. Cho, and B. Hong, "Indexing moving objects for trajectory retrieval on location-based services," *IEICE - Trans. Inf. Syst.*, vol. E90-D, pp. 1388–1397, September 2007.

[6] M. F. Mokbel, T. M. Ghanem, and W. G. Aref, "Spatio-temporal access methods," *IEEE Data Eng. Bull.*, vol. 26, no. 2, pp. 40–49, 2003.

[7] Y. K. Huang and L. F. Lin, "Continuous within query in road networks," in *7th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2011, pp. 1176 – 1181.

[8] Y. Gao, B. Zheng, G. Chen, Q. Li, C. Chen, and G. Chen, "Efficient mutual nearest neighbor query processing for moving object trajectories," *Inf. Sci.*, vol. 180, pp. 2176–2195, June 2010. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2010.02.010

[9] S. Shang, B. Yuan, K. Deng, K. Xie, K. Zheng, and X. Zhou, "PNN query processing on compressed trajectories," *GeoInformatica*, pp. 1–30, Oct. 2011. [Online]. Available: http://dx.doi.org/10.1007/s10707-011-0144-5

[10] Z. Li, Y. Gao, and Y. Lu, "Continuous obstructed range queries in spatio-temporal databases," in *System Science, Engineering Design and Manufacturing Informatization (ICSEM), 2011 International Conference on*, vol. 2, oct. 2011, pp. 267 –270.

[11] L.-V. Nguyen-Dinh, W. G. Aref, and M. F. Mokbel, "Spatio-temporal access methods: Part 2 (2003 - 2010)," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 46–55, 2010.

[12] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, *R-Trees: Theory and Applications (Advanced Information and Knowledge Processing)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[13] 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.

[14] D. Papadias and Y. Theodoridis, "Spatial relations, minimum bounding rectangles, and spatial data structures," *International Journal of Geographical Information Science*, vol. 11, no. 2, pp. 111–138, 1997.

[15] Y. Theodoridis, M. Vazirgiannis, and T. Sellis, "Spatio-temporal indexing for large multimedia applications," in *Proceedings of the 3rd IEEE International Conference on Multimedia Computing and Systems*, 1996, p. 441 448.

[16] E. Frentzos, "Indexing objects moving on fixed networks," in *8th International Symposium on Advances in Spatial and Temporal Databases (SSTD 2003)*. Springer, 2003, pp. 289–305.

[17] V. T. De Almeida and R. H. Güting, "Indexing the trajectories of moving objects in networks*," *Geoinformatica*, vol. 9, pp. 33–60, March 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1046957.1046970

[18] I. S. Popa, K. Zeitouni, V. Oria, D. Barth, and S. Vial, "Parinet: A tunable access method for in-network trajectories," in *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, F. Li, M. M. Moro, S. Ghandeharizadeh, J. R. Haritsa, G. Weikum, M. J. Carey, F. Casati, E. Y. Chang, I. Manolescu, S. Mehrotra, U. Dayal, and V. J. Tsotras, Eds. IEEE, 2010, pp. 177–188.

[19] I. Sandu Popa, K. Zeitouni, V. Oria, D. Barth, and S. Vial, "Indexing in-network trajectory flows," *The VLDB Journal*, vol. 20, pp. 643–669, Oct. 2011.

[20] 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.

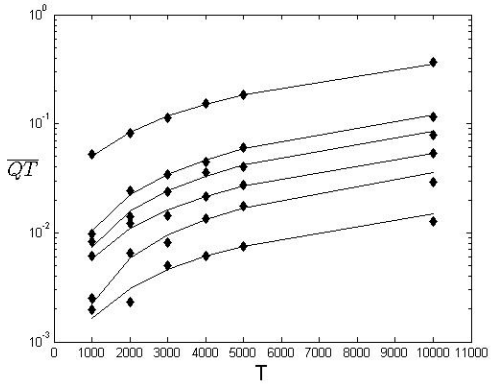
[21] V. P. Chakka, A. Everspaugh, and J. M. Patel, "Indexing large trajectory data sets with seti," in *CIDR*, 2003.

[22] 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.

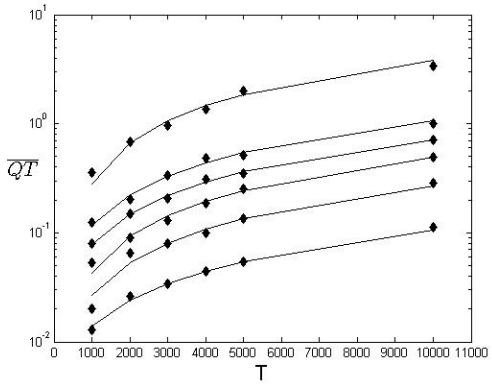
[23] R. Obe and L. Hsu, *PostGIS in Action*. Greenwich, CT, USA: Manning Publications Co., 2011.

[24] PostgreSQL Global Development Group, "PostgreSQL," Accessed: 12/19/2012. [Online]. Available: http://www.postgresql.org/

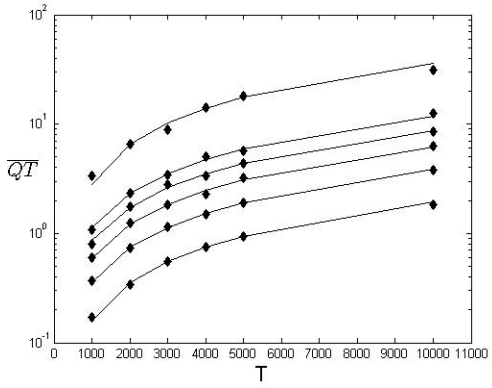
[25] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice in C (2nd Edition)*. Addison-Wesley, 2004.



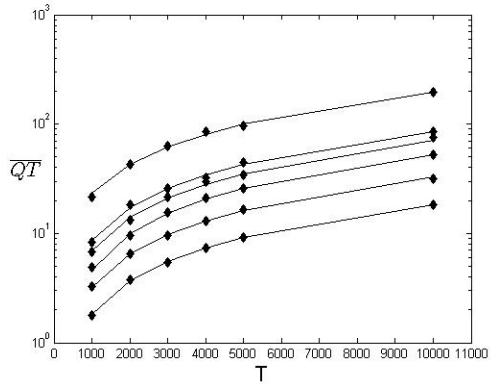
$D_c = 1\%$



$D_c = 5\%$

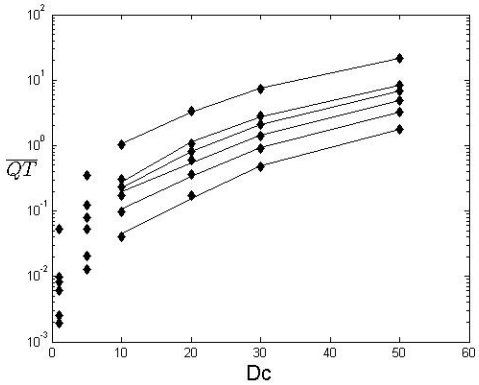


$D_c = 20\%$

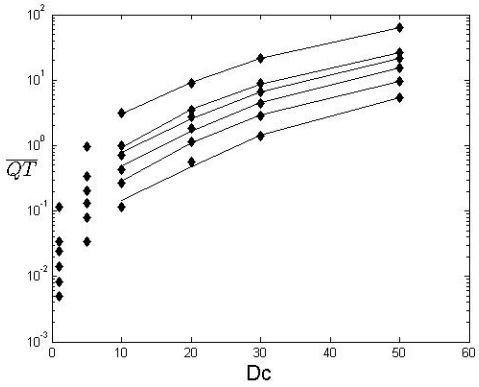


$D_c = 50\%$

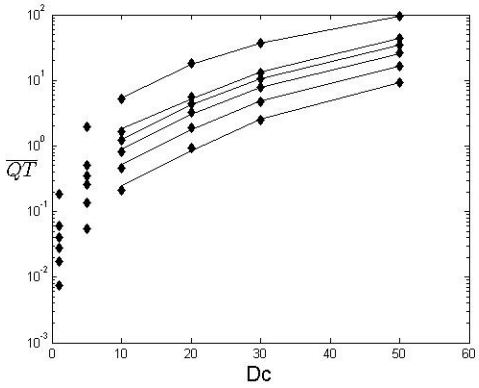
Figure 11. $\overline{QT}$ (in seconds) as the number of trajectories increases having the number of points in each trajectory as parameter.



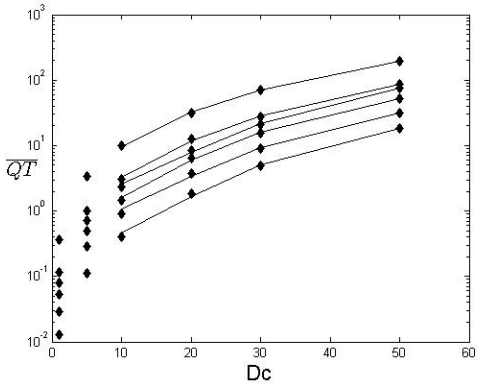
$T = 1000$



$T = 3000$

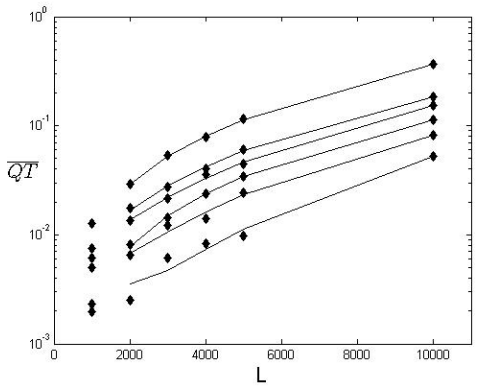


$T = 5000$

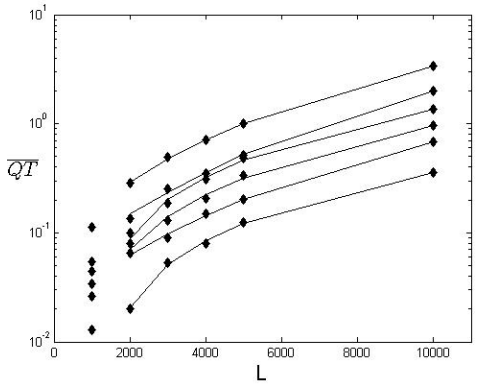


$T = 10000$

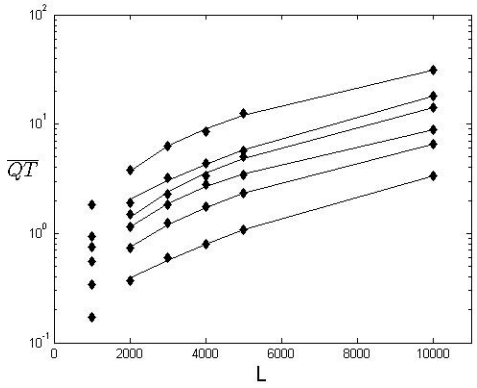
Figure 12. $\overline{QT}$ (in seconds) as the dimension of the querying cube (in percentage of the whole volume) increases and having L as parameter.



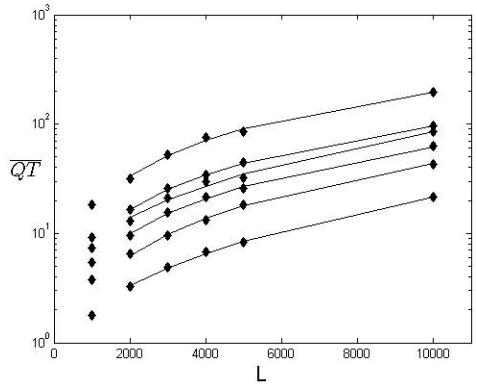
$D_c = 1\%$



$D_c = 5\%$

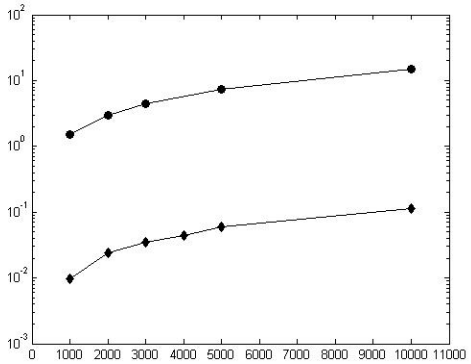


$D_c = 20\%$

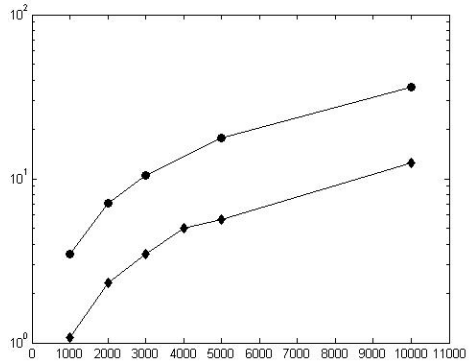


$D_c = 50\%$

Figure 13. $\overline{QT}$ (in seconds) as the number of points in each trajectory increases and having the number of trajectories as parameter.



$D_c = 1\%$



$D_c = 20\%$

Figure 14. The results obtained with the solution proposed in [1] (circles) compared with the results obtained with the solution here as T varies (L=5000).