Efficient Extraction of Motion Flow Data From a Repository of Three-Dimensional Trajectories Using Bi-Dimensional Indexes

Antonio d'Acierno*, Marco Leone†, Alessia Saggese† and Mario Vento†
*Institute of Food Science, National Research Council, Avellino, Italy
† Department of Electronic and Information Engineering (DIEII), University of Salerno, Italy
Email: dacierno.a@isa.cnr.it, {mleone,asaggese,mvento}@unisa.it

***Abstract*—Motivated by the growing presence of acquisition peripherals throughout the world, we propose a novel method for storing and querying moving objects' trajectories extracted from surveillance cameras. Once moving objects have been detected and tracked, we suggest to store and index the related spatio-temporal data by using an innovative scheme based on widely available bidimensional indexes; moreover, a segmentation stage is performed to increase the overall efficiency. Thus, starting from the limitations of most of the clustering and similarity-based approaches, which restrict the choice of the query parameters, we present a trajectory storing system which efficiently supports Dynamic Spatio-Temporal (DST) queries, which are unrestricted time interval queries over moving objects. For the statistical description of the motion flow in the scene, we use a novel query typology, namely the Flow-DST, that is formulated as a sequence of DST. The experimental results, conducted over real and synthetic data, show the efficiency of the approach.**

Keywords-information retrieval; spatio-temporal queries; indexing; segmentation.

I. INTRODUCTION

The presence of monitoring cameras in public and private areas has grown significantly in the last decades. In fact, besides the increasing need for security, more and more public exercises are also interested in using the information extracted from these video sequences for commercial purposes. Think, as an example, to a hypermarket which would like to improve the marketing posters arrangement on the basis of the customers' preferences: a system able to analyze the customers' trajectories and infer their commercial preferences would serve the purpose.

This topic has been recently addressed by a lot of authors. Some examples of real applications able to analyze moving objects' trajectories for commercial purposes are proposed, for instance, in [1] and [2]: in the former, a real hypermarket case study is presented which investigates the relation between daily necessity products and higher flow pattern; in the latter, the authors use laser range finder and cameras to analyze pedestrian behavior in a large shopping mall.

Similar systems, able to perform the analysis of moving objects' trajectories, are far from being simple. In fact, all the sub-systems in which the problem can be decomposed are characterized by its intrinsic challenging issues. In general, the architecture can be summarized as composed by (at least) three main components:

- a *Detection and Tracking Module* that, starting from the acquired video sequence, detects the objects moving in the scene and extracts their trajectories;
- a *Storage Module*, which is in charge of storing the extracted data by means of suited indexing strategies;
- a *Retrieval Module*, which allows to retrieve salient data for visualization and statistical purposes on the basis of the specific queries submitted by the user.

As for the first of the above mentioned phases, that is the extraction of the motion trajectories of moving objects, different tracking algorithms have been proposed [3][4], providing reasonably usable solutions.

On the other hand, only a modest attention has been devoted to systems for storing and retrieving information from very large scale Moving Object Databases (MODs) [5]. In fact, the major part of the works dealing with the analysis of motion information has mainly focused on clustering [6] or anomalous detection [7]; even some of these approaches are particularly efficient, they suffer from a common limitation: these methods only allow pre-determined queries, which involves the use of devised and optimized system architecture for supporting a bunch of queries referring to a given spatial area. It also means that the user is not allowed to choose the query parameters at query time, but he can only fix the retrieval parameters in the pre-processing phase.

While it is simple to imagine how much the flexibility degree of such a type of system can increase, it is not likewise to design a system architecture having these potentiality.

The most significant contributions to design an architecture able to cope with large amount of trajectory data can be obtained by browsing the literature coming from the database field. A widely adopted solution for spatial indexing founds on R-trees [8], which are tree data structures that hierarchically organize geometric bidimensional data by representing each object through its Minimum Bounding Rectangle (MBR). Starting from Guttman's pioneering paper, a lot of spatio-temporal indexing scheme have been proposed for many applications contexts, most of which are optimizations of R-trees [9][10].

 ISBN: 978-1-61208-227-1

In order to overcome the above mentioned limitations of the most common trajectories analysis systems by taking advantage from the solution designed in the database field, we propose a general-purpose system for the analysis of moving objects' trajectories which allows the user to choose at query time the query parameters. The generic scenario can be briefly described as follows: a camera records the moving objects in the entire scene for a given, generally long, period of time, but no area of interest is a priori defined. The system finds and efficiently stores the objects' trajectories and allows to solve Dynamic Spatio-Temporal (*DST*) queries, e.g. queries finding all the trajectories passing through an area defined directly by the user within the query (i.e. at query time). Moreover, the system also supports Flow-DST queries, which provide complex statistical analysis in a fully configurable format. Our system extends an off-the-shelf solution for storing the collected data; in particular, it handles with the problem that the actual spatial indexes for three-dimensional data are not widely available and extends a redundant solution we proposed recently [11].

II. The Proposed Method

A generic trajectory T^i can be expressed as a sequence of spatio-temporal points $T^i =< P_1^i, P_2^i, ..., P_N^i >$, where the generic point $P_k^i = (x_k^i, y_k^i, t_k)$ represents the position (x_k^i, y_k^i) of the i-th object at time t_k. We choose to represent each trajectory T^i according to the line segments model, so that a trajectory is obtained by interpolating consecutive points of the sequence. Furthermore, we associate to each trajectory its relative *Minimum Bounding Rectangle (MBR)*, corresponding to its maximum extents in the three-dimensional space.

Once proper indexing strategies have been performed, the problem becomes to find fast and effective means for information extraction from the stored data; for this purpose, we introduce two novel query types, which have been made available in the system; the former, namely the Dynamic Spatio-Temporal (*DST*) query, is the basis for the formulation of more complex queries and allows to answer requests from the user searching for the objects' trajectories passing through a given spatial area in a given time interval.

More formally, a typical *DST* query would appear as: "find all the people passing through a given area A_1 in a given time interval $[t_s, t_e]$"; according to this formulation, we can think to the area as a rectangle with coordinates (x_s, y_s) and (x_e, y_e) while $[t_s, t_e]$ are the starting and final time instants. This leads to the definition of a query box B, which can be associated to each *DST* query:

$$B = \{(x_s, y_s, t_s), (x_e, y_e, t_e)\},$$

The spatio-temporal volume is geometrically defined by a volume delimited by its bottom-left-back point (x_s, y_s, t_s) and its top-right-front point (x_e, y_e, t_e); this volume, in turn, is composed by a spatial interval with top-left point (x_s, y_s) and bottom-right point (x_e, y_e) in a bidimensional space, and a temporal interval (t_s, t_e) representing the third coordinate. An instance of a *DST* query is shown in Figure 1a.

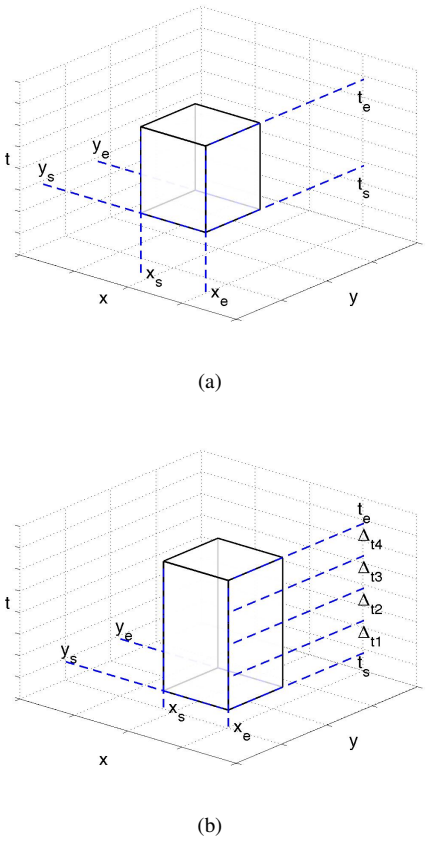

Figure 1: Geometric interpretation of different types of query: *DST* (a) and *F-DST* (b).

Starting from the formalization of the *DST* query, we propose a specialization of it, namely the Flow-DST *(Flow-DST)*, useful for real commercial applications. The Flow-DST *(F-DST)* query is introduced for analyzing the objects' motion flow in the observed scene; typical examples of this query typology are *"find the total number of vehicles passing by a given street (the spatial area can be dynamically defined at query time) each week-end of the last three months"*, or *"find the number of pedestrians passing by a given access gate each couple of hours during the last two days"*. From a geometric perspective, a *F-DST* can be seen as the application of many DST queries so as to obtain results at fixed time intervals, as shown in Figure 1b.

A. Indexing and Query Answering

In this subsection, we will describe the proposed indexing scheme with reference to the *DST* query since, as it has been clearly described in the query formalization section, the *F-DST* can be derived from it.

 ISBN: 978-1-61208-227-1

A simple algorithm for retrieving the trajectories satisfying a *DST* 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 [12] is needed. Unfortunately, despite of its simplicity, the use of a clipping algorithm is not suited for handling large datasets, so demanding for more efficient approaches. This is the main motivation why we propose to use spatial indexing strategies to reduce both the time needed to extract the trajectories from the database and the number of trajectories to be clipped; this leads, as a consequence, to the real-time processing of a *DST* query. Before going into detail about the use of these indexes, we here recall some basic aspects of spatial indexing.

Spatial indexes are usually aimed to improve the efficiency when handling with geometric data types like points, lines and polygons and querying spatial relationships among them. Although many commercial databases provide efficient three-dimensional indexes, these usually restrict the intersection operation to the bidimensional case; for this reason, we propose to represent the 3D problem in terms of one or more 2D sub-problems. While this choice allows to take advantage of off-the-shelf 2D solutions, it must be noticed that, in the bidimensional space, the intersection between the trajectory and the corresponding query box is a necessary but not sufficient condition; in fact, when the trajectory intersection with the query box holds in each of the three 2D planes, it will not necessarily hold in the 3D plane too, while the opposite is trivially true.

Starting from the above considerations, we represent the *i-th* trajectory T^i through the original sequence of points in the 3D space (x, y, t), together with the three different MBR projections (MBR_{xy}, MBR_{xt} and MBR_{yt}), as shown in Figure 2.a and 2.b.

We verify the intersection of the trajectories with the corresponding bidimensional query box in each of the three 2D planes, as depicted in Figure 2.c. Let I_{xy}, I_{xt} and I_{yt} be the resulting sets of trajectories intersecting the query box and defined as:

$$I_{xy} = \{T : MBR_{xy}(T) \cap B_{xy} \neq \emptyset\} \quad (1)$$

$$I_{xt} = \{T : MBR_{xt}(T) \cap B_{xt} \neq \emptyset\} \quad (2)$$

$$I_{yt} = \{T : MBR_{yt}(T) \cap B_{yt} \neq \emptyset\}, \quad (3)$$

where B_{xy}, B_{xt} and B_{yt} are the three projections of the 3D query box B. The set C of candidate trajectories to be clipped in the 3D space is therefore defined as $C = \{I_{xy} \cap I_{xt} \cap I_{yt}\}$. At last, we apply the clipping algorithm and obtain the final intersection result, as shown in Figure 2.d.

According to the indexing strategy above described, the capability to significantly reduce the number of trajectories to be clipped plays a crucial role, as the huge amount of trajectory data represents a key factor of complexity. As a consequence, we introduce a segmentation stage aimed at improving the *selectivity* of the indexes, which, in turn, only depends on the trajectory geometry.

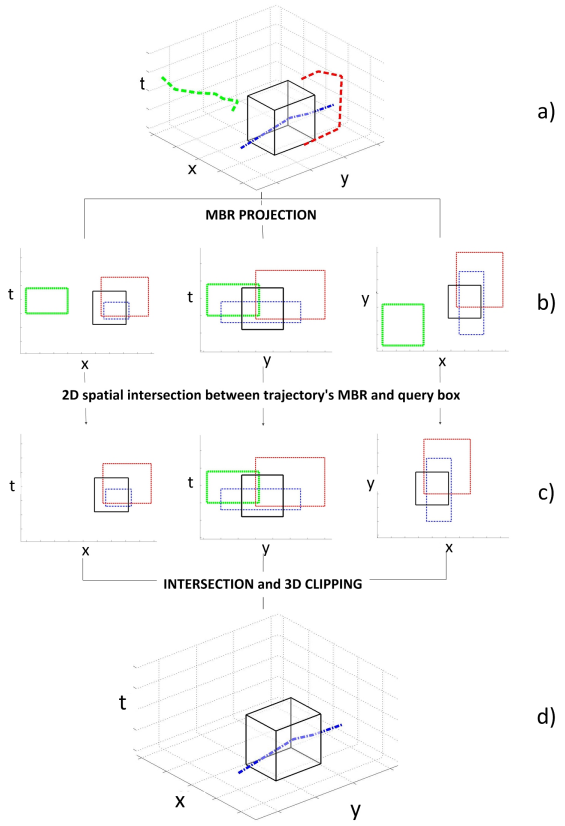

Figure 2: An overview of the method. (a) a query box and three trajectories; (b) the projections of the trajectory MBRs on the planes; (c) the MBR projections intersecting the query box (the three MBR projections of the red and blue trajectories intersect the query box); (d) the final result of our method, after the application of the clipping algorithm.

B. Segmentation

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.

The proposed algorithm works recursively: initially (that is at iteration 0) it assumes that the trajectory T^k is composed by a single unit ${}^0U_1^k$, that is split into a set of m consecutive smaller units $\{{}^1U_1^k, \ldots, {}^1U_m^k\}$; each of the ${}^1U_i^k$ is in turn inspected and, if the stop criteria are not satisfied, it is further split.

Let us analyze how a generic unit ${}^{(i-1)}U = \{P_1, \ldots, P_m\}$ at iteration $i-1$ is split into $\{{}^iU_1, \ldots, {}^iU_n\}$ at iteration i; 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, ${}^iU_1$ is the set of the consecutive points lying on the left of the *split-value* ${}^iU_1 = \{P_1, \ldots, P_k\}$, 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*, 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. Let V be the volume containing all the trajectories stored until this moment. 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's MBRs, with respect to the projections of V on the coordinate planes (V_{xy}, V_{xt} and V_{yt}):

$$O_{xy} = \frac{MBR_{xy}(U)}{V_{xy}} \tag{4}$$

$$O_{xt} = \frac{MBR_{xt}(U)}{V_{xt}} \tag{5}$$

$$O_{yt} = \frac{MBR_{yt}(U)}{V_{yt}} \tag{6}$$

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.

The regular termination of the algorithm is reached when all the trajectory points have been processed; anyway, an abnormal termination is also possible during each iteration step, on the basis of two stop criteria: PS^{min} attains the minimum number of points belonging to a trajectory unit, while PA^{min} is the minimum allowed size, in percentage value with respect to the entire scenario, of an MBR area.

III. Experimental Results

The system has been implemented by storing the trajectory data in Postgres using PostGIS [13], being the latter Postgres extension for storing spatial data like points, lines and polygons. Data are indexed using the standard bidimensional R-tree over GiST (Generalized Search Trees) indexes; the specialized literature highlights that this choice guarantees higher performance in case of spatial queries, if compared with the PostGIS implementation of R-trees. Once data have been indexed, PostGIS provides a very efficient function to perform intersections between boxes and MBRs in a 2D space.

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 the PostrgreSQL 9.1 server and the 1.5 version of PostGIS. We tested our retrieval system with real and synthetic data. The synthetic data 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. Table I reports the defined free parameters with the values used to generate our data.

Table I: THE PARAMETERS USED IN OUR EXPERIMENTS.

Scene width (pixels)	10^4
Scene height (pixels)	10^4
Time interval length (secs)	10^5
PI_x	5%
PI_y	5%
V_x^{max}	10 pixels/secs
V_y^{max}	10 pixels/secs

First, we decided to test our segmentation algorithm assuming $PA^{min} = 0.1\%$ and $PS^{min} = 100$; 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). Last, the obtained N_{seg}^i are averaged over L, so obtaining $\overline{N_{seg}}$. Not surprisingly, we have, with very good approximation, that the number of segments $\overline{N_{seg}}$ linearly increases with the trajectory length L, as shown in Figure 3.

The use of the segmentation algorithm clearly makes the proposed system able to outperform the method presented in [11], as depicted in Figure 5. For the sake of readability, Figure 5 only highlights the improvement for $D_c \in \{10\%, 50\%\}$ and having $L = 5000$. In particular, the diamonds refer to the new method, while the circles refer to the previous one.

Furthermore, we investigate on the time needed to process a Flow-DST query (FT). We can note that FT is N times (N being the number of intervals we are interested in) the

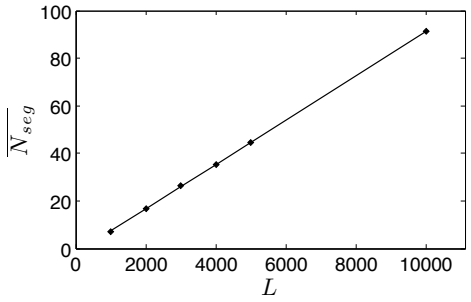

Figure 3: The performance of the segmentation algorithm.

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

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

time needed to perform a DST query (QT); QT, in turn, is a function of at least four 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:

$$FT = N * f(T, L, D_c, P_c). \quad (7)$$

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

$$\overline{FT} = f(T, L, D_c, N). \quad (8)$$

Furthermore, we have experimentally verified that $\overline{N_{seg}}/L = k$, with k constant (Figure 3). It means that, on average, the length of each unit can be assumed to be constant. We can thus derive, with good approximation, that:

$$\overline{FT} = N * f(\overline{N_{seg}}, D_c). \quad (9)$$

Starting from these considerations and assuming $L = 1000$, we can measure the time needed to perform several Flow-DST queries with $D_c \in \{5\%, 10\%, 20\%, 30\%, 50\%\}$ as N_{seg} varies. Figure 4 shows the obtained results. It can be noticed that $\overline{N_{seg}}$ linearly increases with f. However, it is worth pointing out that the time needed to process each query is significantly influenced by the clipping procedure which, in turn, is strongly dependent on the time for the extraction of each trajectory: in fact, a segment unit need to be extract before to be clipped. It clearly means that an optimization of the segmentation parameters can still improve the performance of our method.

Table III: AVERAGED TIME (IN SECONDS) TO SOLVE A *DST* QUERY ON THE MIT TRAJECTORIES DATASET.

D_c	N	T^1	T^2	T^3	$\overline{QT}$
1%	200	0.003	0.010	0.009	0.022
5%	40	0.007	0.064	0.115	0.186
10%	20	0.013	0.154	0.320	0.487
20%	10	0.038	0.533	1.383	1.954
30%	7	0.097	1.566	4.014	5.673
50%	4	0.173	5.878	14.924	20.975

It must be noticed that the synthetic data really stressed the system, resulting in trajectories with tens of millions of points, which is over and above the average trajectory length of available datasets. To confirm this consideration, we also tested the performance of our indexing scheme on a well-known real dataset, the freely available MIT trajectory dataset [14], obtained from a parking lot scene within five days; the dataset is composed of approximately $4 * 10^4$ trajectories with 108.81 points in each trajectory (on average). At loading time, each trajectory has been segmented using $PA^{min} = 1$ and $PS^{min} = 100$, so obtaining approximately $1.92 * 10^6$ segments with 23.71 points in each segment (on average). Table III shows $\overline{QT}$ (in seconds) as D_c varies. The table also shows $\overline{QT}$ results from the sum of three terms: T^1 is the time needed to select the segments whose bounding boxes intersect the query box on each bi-dimensional plane, T^2 is the time to clip the segments while T^3 is the time needed to extract the whole trajectory. It is possible to note that the obtained results confirm the efficiency of the proposed method.

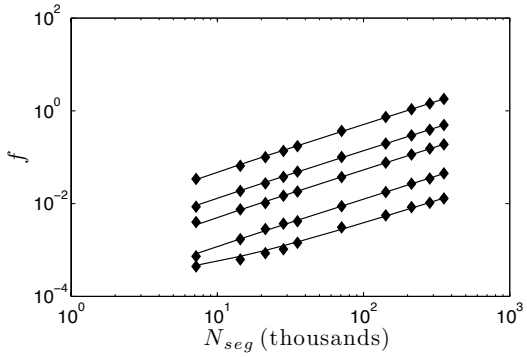

Figure 4: The performance of the system as N_{seg} vary for different values of D_c.

IV. CONCLUSION

We addressed the problem of efficiently storing, indexing and querying spatio-temporal data for motion trajectory analysis. Our main contributions lie in a significant improvement of the overall efficiency and the wide adaptability of the queries. The former contribution is achieved by proposing an

 ISBN: 978-1-61208-227-1

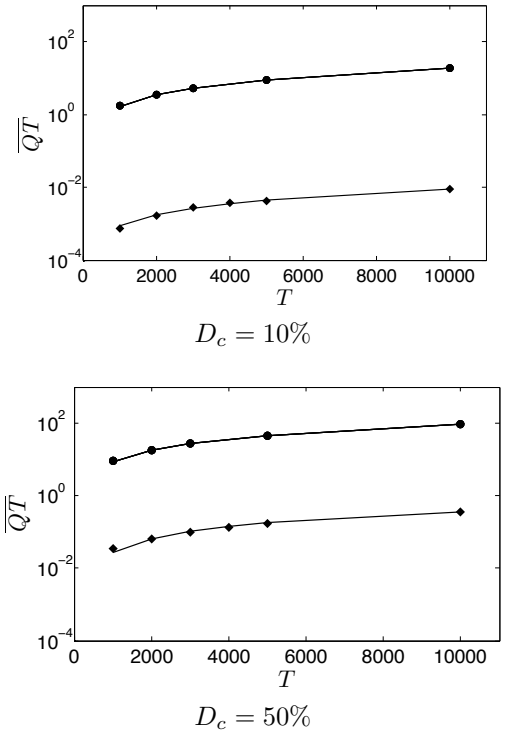

Figure 5: The results obtained with the solution proposed in [11] (circles) compared with the results obtained with the solution here as T varies (L=5000).

indexing scheme based on the use of off-the-shelf solutions; in addition, the introduction of *Dynamic Spatio-Temporal* and *Flow DST (F-DST)* queries has provided means for defining the query parameters at runtime and also for answering frequently occurring retrieval problems.

The preliminary tests, conducted over synthetic and real data, confirm the effectiveness of the approach in terms of query generality and computational efficiency. Anyway, some improvements are still possible. First, the application of multithreading for the clipping algorithm could significantly improve the performance of our method, since the system could take advantage from multi-core and multi-processors systems. In addition, a deeper analysis could be conducted on the optimal choice of the segmentation parameters.

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 FLAGSHIP *InterOmics* project (PB.P05, funded and supported by the Italian MIUR and CNR organizations).

REFERENCES

[1] K. Teknomo and P. G. Gerilla, "Pedestrian Static Trajectory Analysis of a Hypermarket," in *Proceedings of the 8th International conference of the Eastern Asia Society for Transportation Studies*, vol. 7, Nov. 2009, pp. 1–12.

[2] K. Okamoto, A. Utsumi, T. Ikeda, and H. Yamazoe, "Classification of pedestrian behavior in a shopping mall based on lrf and camera observations," in *IAPR Conference on Machine Vision Applications*, vol. 110, no. 421, 2011, pp. 233–238.

[3] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, no. 4, p. 13, Dec. 2006.

[4] R. DiLascio, 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, pp. 104–113.

[5] 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 *Proceedings of the "IEEE Conference on Intelligent Transportation Systems (ITSC)"*, 2012, pp. 989–994.

[6] B. T. Morris and M. M. Trivedi, "Learning Trajectory Patterns by Clustering: Experimental Studies and Comparative Evaluation," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2009, pp. 312–319.

[7] G. Acampora, P. Foggia, A. Saggese, and M. Vento, "Combining neural networks and fuzzy systems for human behavior understanding," in *Proceedings of the "IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)"*, 2012, pp. 88–93.

[8] 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.

[9] Z. Song and N. Roussopoulos, "Seb-tree: An approach to index continuously moving objects," in *Proceedings of the 4th Conference on MDM*. London, UK: Springer-Verlag, 2003, pp. 340–344.

[10] J. Priyadarshini, P. AnandhaKumar, M. Aparna, J. Geetha, and N. Shobana, "Indexing and querying technique for dynamic location updates using r k-d trajectory trie tree," in *International Conference on Recent Trends in Information Technology (ICRTIT)*, 2011, pp. 1143 –1148.

[11] A. d'Acierno, A. Saggese, and M. Vento, "A redundant bi-dimensional indexing scheme for three-dimensional trajectories," in *Proceedings of the 1th Conference on Advances in Information Mining and Management*, 2011, pp. 73–78.

[12] 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.

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

[14] X. Wang, K. T. Ma, G.-W. Ng, and W. E. Grimson, "Trajectory analysis and semantic region modeling using nonparametric hierarchical bayesian models," *Int. J. Comput. Vision*, vol. 95, pp. 287–312, December 2011.

 ISBN: 978-1-61208-227-1