IX Brazilian Symposium on GeoInformatics, Campos do Jordão, Brazil, November 25-28, 2007, INPE, p. 61-72.

Trajectory Data Warehouses: Proposal of Design and Application to Exploit Data

Fernando J. Braz[1]

[1]Department of Computer Science – Ca' Foscari University - Venice - Italy

fbraz@dsi.unive.it

Abstract. *In this paper we are interested in storing and perform OLAP queries about various aggregate trajectory properties. We consider a data stream environment where a set of mobile objects send the data about its location in a irregular and unbounded way, the data volume is stored in a centralized and traditional DW with pre-computed aggregations values (preserving the trajectories privacy). We present an application developed to receive the stream data set, to store and compute the pre-aggregation values and to present final results in order to reveal the knowledge about the trajectories.*

1. Introduction

The data warehouse traditional model can be resumed as a subject-oriented data collection integrated from various operational databases. On a data warehouse model, the data are summarized and aggregated in a multidimensional way in order to facilitate access and data analysis. A Data warehouse provides an integrated environment by extracting, filtering, and integrating relevant information from various data sources. A Data Stream environment presents some characteristics that may improve the difficulty to build and maintain a data warehouse. The data arriving on an unpredictable and continuous rate, the larger data volume and the resource constraints (memory, processing) are some of these main characteristics. The data warehouse traditional model must be adapted in order to work in agreement with these constraints.

The development of new technologies for mobile devices and low-cost sensors results in the possibility of storing larger data volumes about trajectories of moving objects. These data volumes could be stored on a multidimensional model in order to allow an accuracy analysis, it can be defined as a trajectory data warehouse. The goal is to store, manage and analyze the trajectories data in a multidimensional way. The trajectory can be represented by position (X and Y) and time data. A set of observations represents data about several moving objects positions. The trajectory data warehouse has two main problems: the loading phase and the computing of measures. The trajectory data of moving objects arrive in an unbounded and unpredictable way, it is a characteristic that must be considered in the building of a multidimensional data warehouse model. The loading phase has to receive and process the data volume considering the available resources and the irregular rate of arriving the data. We consider a data warehouse model where the identifier of the trajectories is abstracted in favour of aggregate information concerning global properties of a set of moving objects. The aggregated information stored in each cell of the DW model can be used to reveal knowledge of the objects. It can be done by the usage of the OLAP operators, these results can be used as input for subsequent analyses.

In this paper we present an application developed to receive the stream data set, to store and compute the pre-aggregation values and to present final results in order to reveal the knowledge about the trajectories. This application works in a data stream environment, it can receive the data stream volumes (*loading phase*), compute and store the aggregation values (*computing measures phase*) in a trajectory data warehouse. The application was built considering the model proposed in [Braz et al. 2007], we have used a traditional DW system in order to store the Trajectory DW. In the Section 2 we present a briefly review about Trajectory DW model, the main problems and the DW model used in order to store the trajectories are also presented. The application is detailed in the Section 3. Finally, in the Section 4 we draw some conclusions and possible future research topics.

2. The Trajectory DW Model

There are some proposals of spatial data warehouses [Han et al. 1998],[Rivest et al. 2001],[Marchant et al. 2004],[Shekhar et al. 2001], but none of these proposals work with objects moving in a continuous way in time. However, in the building of a warehouse for trajectories, it is a crucial issue. The movement of a spatio-temporal object *o* - i.e., it *trajectory* - can be represented by a finite set of *observations*,i.e. a finite subset of points taken from the actual continuous trajectory. This finite set is called a *sampling*.

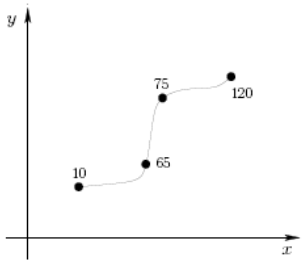

Figure 1. Trajectory with a sampling

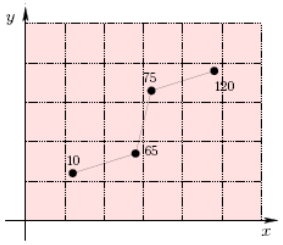

Figure 2. Linear interpolation

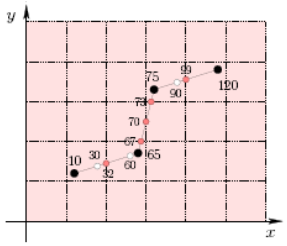

Figure 3. Interpolated trajectory with spatial and temporal points

The Figure 1 represents a trajectory of a moving object in a two dimensional space. Each point of the trajectory is represented by a tuple *(id, x, y, t)* corresponding to an object *id* in a location *(x, y)* at time *t*. There are some situations where we have to reconstruct the trajectory of the moving object from its sampling, e.g., when one is interested in computing the cumulative number of trajectories in a give area. In [Braz et al. 2007] the proposal is to use *linear local interpolation* in order to do it, assuming the movement of the objects between the observed points happens with constant speed, in a straight way. A Trajectory Data Warehouse (*TDW*) has to capable to store a stream of samplings, process the data volume, compute and store the measures in order to provide an environment to analyze the information about the objects. The aggregations measures are crucial in order to do it. However, in a data stream environment, where the data arrive in an irregular and unpredictable way it is specially difficulty. In the *loading phase* the available resources are a very important constraint, it is necessary to develop some mechanism in order to limit the consumption of the resources and to improve the performance of the process. Besides, there is another important phase: *computing measures*, several measures can be

computed in agreement with different complexity spaces. Therefore, the *TDW* must be modeled considering all these issues. In this application we have used the model proposed in [Braz et al. 2007].

2.1. Loading Problem

The loading begins at the base cells of the base cuboid, with suitable sub-aggregate measures, from which starting to compute super-aggregated functions. Considering the data stream characteristic, we have to limit the amount of buffer memory needed to store information about active trajectories. In agreement with [Braz et al. 2007] we consider a trajectory ended when, for a long time interval, no further observation has been received. Given the observations for a trajectory shown in the Figure 1, a possible reconstructed trajectory using linear interpolation is shown by the Figure 2, where we also illustrate the discretized 2D space. With the linear interpolation is possible to infer additional spatio-temporal locations of intermediate points, these points occur between two known trajectory observations. Updating the data warehouse on the basis of each single observation, the *measures* ($M_1, \ldots, M_k$), possibly corresponding to the four observations of our example, the Table 1 shows the base cells.

Table 1. Cells representation - for each observation

Time label	X	Y	T	M_1	...	M_k
10	[30,60)	[30,60)	[0,30)	...	...	...
65	[60,90)	[30,60)	[60,90)	...	...	...
75	[90,120)	[90,120)	[60,90)	...	...	...
120	[120,150)	[90,120)	[60,120)	...	...	...

Table 2. Sequence of segments composing the interpolated trajectory, and the base cells that completely include each segment.

($\mathbf{t}_i$, $\mathbf{t}_{i+1}$)	X	Y	T	M_1	...	M_k
(10,30)	[30,60)	[30,60)	[0,30)	...	...	...
(30,32)	[30,60)	[30,60)	[30,60)	...	...	...
(32,60)	[90,120)	[30,60)	[30,60)	...	...	...
(60,65)	[90,120)	[30,60)	[60,90)	...	...	...
(65,67)	[90,120)	[30,60)	[60,90)	...	...	...
(67,70)	[90,120)	[90,120)	[60,90)	...	...	...
(70,73)	[120,150)	[90,120)	[60,90)	...	...	...
(73,75)	[120,150)	[120,150)	[60,90)	...	...	...
(75,90)	[120,150)	[120,150)	[60,90)	...	...	...
(90,99)	[120,150)	[120,150)	[90,120)	...	...	...
(99,120)	[150,180)	[120,150)	[90,120)	...	...	...

Before the interpolation some base cells could be traversed by the trajectories but, since no observation falls in them, they not appear in the *fact* table, the solution proposed in [Braz et al. 2007] is to consider the additional interpolated points for each cell traversed by a trajectory. The interpolation is computed considering the intersections between the trajectory and the border of the spatio-temporal cells. The Figure 3 shows a trajectory

considering the additional interpolated points. The interpolated points, associated with temporal labels 30, 60, and 90, have been added to match the granularity of the temporal dimension. In fact, they correspond to cross points of a temporal border of some 3D cell. The points labeled with 32, 67, 70, 73, and 99, have been instead introduced to match the spatial dimensions.

After the including of these additional interpolated points, we have further 3D base cells in which we can now store significant measures associated with the trajectory of the given object. The new points subdivide the interpolated trajectory into small segments, each one completely included in some 3D base cell. Therefore we can now update a cell measure on the basis of a single trajectory segment. The Table 2 shows the sequence of edges composing the interpolated trajectory of Figure 3, and the base cell which the edge belongs to.

2.2. Aggregation Problem

A typical measure in a trajectory data warehouse can represent any interesting property about the trajectories in a spatio-temporal interval. In [Gray et al. 1997] the authors present a classification of aggregate functions based on the space complexity for computing a super-aggregate starting from a set of sub-aggregates previously computed:

- *Distributive*: The super-aggregates can be computed from the sub-aggregates.
- *Algebraic*: The super-aggregates can be computed from the sub-aggregates together with a finite set of auxiliary measures .
- *Holistic*: The super-aggregates cannot be computed from the sub-aggregates, not even using any finite number of auxiliary measures.

Table 3. Numeric measures

m1	*numobs*	*Number of observations in the cell*
m2	*trajinit*	*Number of trajectories starting in the cell*
m3	*presence*	*Number of trajectories in the cell*
m4	*distance*	*Total distance covered by trajectories in the cell*
m5	*speed*	*Average speed of trajectories in the cell*
m6	v_{max}	*Maximum speed of trajectories in the cell*

The Table 3 shows the measures that are considered in the application. The computation of the super-aggregates for the measures *m1, m2, m4* and *m6* uses *distributive* aggregate functions. After the loading of the base cells with the exact measures is possible to accumulate the measures by usage the function *sum* (*m1, m2* and *m4*) and *max* (*m6*). However, the super-aggregate for the measure *m5* is *algebraic*, it is necessary to compute an auxiliary measure in order to compute the aggregate function. A pair $\langle distance, time \rangle$ must be considered, where *distance* is the measure *m4* and *time* is the total time spent by trajectories in the cell. For a cell *C* arising as the union of adjacent cells, the cumulative function performs a component-wise addition, producing a pair $\langle distance_f, time_f \rangle$, therefore the average speed in *C* is computed by $distance_f / time_f$. The aggregate function for *m3* is *holistic*, then is necessary to compute the measure in an approximated way. In this application we have used the approach presented in [Braz et al. 2007], the approach will be presented in the following.

The *Presence* function represents the count of the *distinct* trajectories crossing a given cell. Since this function has to deal with the issues related to counting *distinct* trajectories, it is a sort of `COUNT_DISTINCT()` aggregate, and thus a *holistic* one. We exploit alternative and *non-holistic* aggregate functions to compute *Presence* values that *approximate* to the exact ones. These alternative functions only need a few/constant memory size for maintaining the information – i.e., the M-tuple – to be associated with each base cell of our data warehouse, from which we can start to compute a super-aggregate. The two approximate functions we will consider are the following:

1. *Presence*$_{Distributive}$: We assume that the only measure associated with each base cell is the exact (or approximate) count of all the *distinct* trajectories crossing the cell. Therefore, the super-aggregate corresponding to a roll-up operation is simply obtained by summing up all the measures associated with cell. This aggregate function may produce very inexact approximation of the true *Presence*. Because we may count multiple times the same trajectory. We do not have enough information in the base cell that permit us to perform a *count distinct* when rolling-up.

2. *Presence*$_{Algebraic}$: Each base cell stores an M-tuple of measures. One of these is the exact (or approximate) count of all the *distinct* trajectories touching the cell. The other measures are used when we compute the super-aggregate, in order to correct the errors introduces by function *Presence*$_{Distributive}$ due to the duplicated count of trajectory presences.
   More formally, let $C_{x,y,t}$ be a generic base cell of our cuboid, where x, y, and t identify intervals of the form $[l, u)$, in which we have subdivided the spatial and temporal dimensions. The associated measures are thus $C_{x,y,t}.presence$, $C_{x,y,t}.crossX$, $C_{x,y,t}.crossY$, and $C_{x,y,t}.crossT$.
   $C_{x,y,t}.presence$ is the count of all the *distinct* trajectories crossing the cell.
   $C_{x,y,t}.crossX$ is the number of *distinct* trajectories crossing the *spatial* border between $C_{x,y,t}$ and $C_{x+1,y,t}$.
   $C_{x,y,t}.crossY$ is the number of *distinct* trajectories crossing the *spatial* border between $C_{x,y,t}$ and $C_{x,y+1,t}$.
   Finally, $C_{x,y,t}.crossT$ is the number of *distinct* trajectories crossing the *temporal* border between $C_{x,y,t}$ and $C_{x,y,t+1}$.
   In order to compute the super-aggregate corresponding to two adjacent cells with respect to a given dimension, namely $C_{x',y',t'} = C_{x,y,t} \cup C_{x+1,y,t}$, we can compute it as follows:

$$\begin{aligned} &\textit{Presence}_{Algebraic}(C_{x,y,t} \cup C_{x+1,y,t}) = \\ &= C_{x',y',t'}.presence = \\ &= C_{x,y,t}.presence \ + \ C_{x+1,y,t}.presence \ - \ C_{x,y,t}.crossX \end{aligned} \tag{1}$$

   Moreover, if we need to update the other measures associated with the $C_{x',y',t'}$ for subsequent aggregations, we have:

$$C_{x',y',t'}.crossX = C_{x+1,y,t}.crossX$$
$$C_{x',y',t'}.crossY = C_{x,y,t}.crossY + C_{x+1,y,t}.crossY$$

$$C_{x',y',t'}.crossT = C_{x,y,t}.crossT + C_{x+1,y,t}.crossT$$

Then, the idea is to compute a *holistic* measure by the usage of the *distributive* and *algebraic* measures. Therefore, the final result of the measure is an approximated value, computed in agreement with the limited resources presented in a data stream environment.

3. The Application

We have developed our application using the synthetic datasets generated by the traffic simulator described in [Brinkhoff 2000]. These data are stored in the Trajectory Data Warehouse (*TDW*) model presented in the Section 2. The measures stored in the *TDW* can be used to discover interesting phenomena of the trajectories. The application tries to solve the both problems: loading and aggregation, which were presented in the Section 2.1 and 2.2.

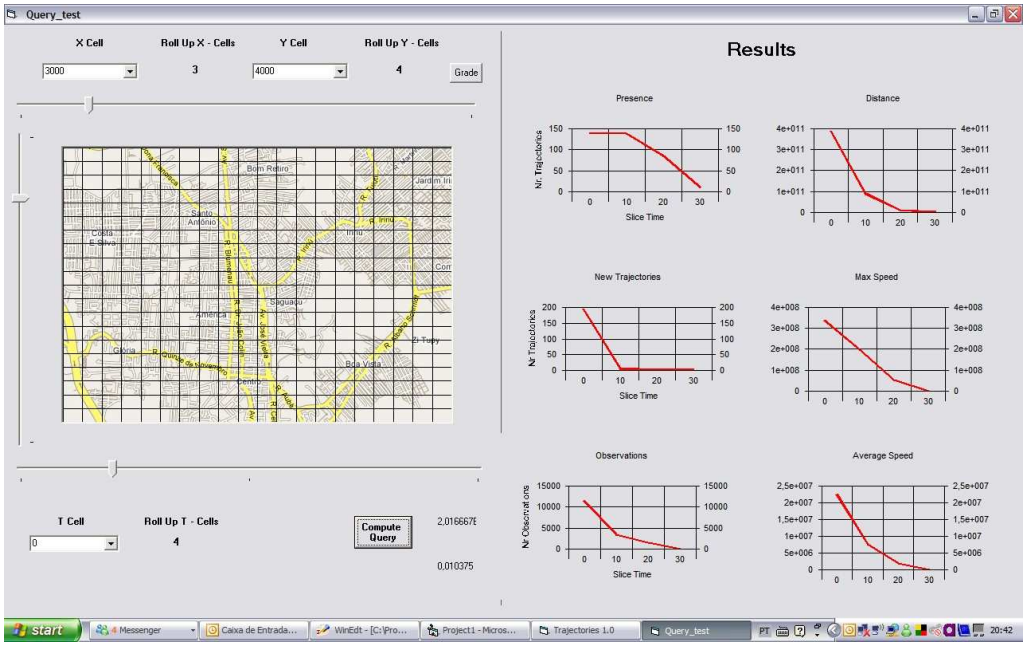

Figure 4. Results Interface

The Figure 4 shows the interface where is possible to visualize the values of the measures computed considering a cell selected by the user. The result presents the evolution of the values of measures in a range of time, in the same visualization is possible to define different values of roll-up operations. The roll-up operation can be defined by the usage of the slider controls over the map, a more detailed explanation will be presented in the next sections.

The *TDW* was implemented in a traditional data warehouse tool, we have used the *MS SQL SERVER 2005*. The *TDW* was modeled in agreement with the *star* model [Kimball 1996], with a fact table and three dimension tables (*X* and *Y* spatial dimensions and *T* temporal dimension). The structure of these tables can be found in the Tables 4, 5 and 5. The Figure 5 shows a schema of the our application: a bottom level where is the *TDW* and the *buffer table*; and a first level where works the loading and aggregation

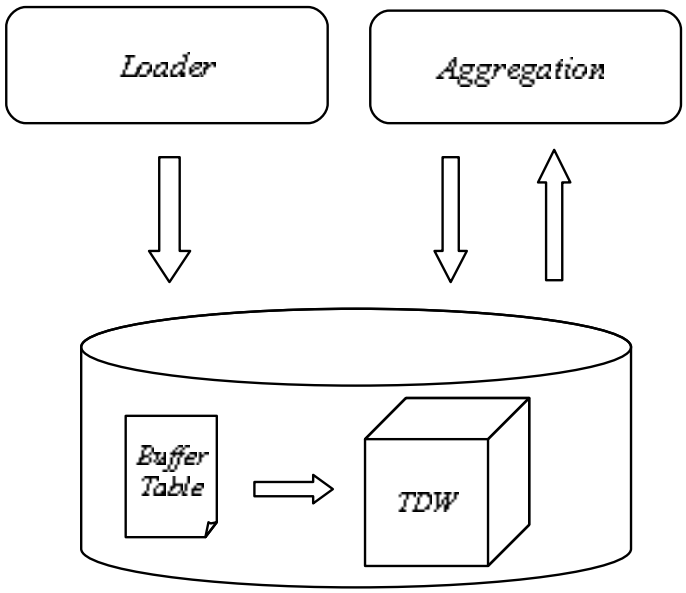

Figure 5. Application Schema

components. The arrows represent the data communication among the components, these components will be described in following sections. The application was built with the *Visual Basic* language, the application allows the user to access the *DTW*. A lot of settings can be done: the definition of the level of the granularity of the trajectory data warehouse, to control the loading of the *TDW* and computation of the measures and aggregations are some of the possibilities of the usage.

Table 4. Fact Table

tid	*time foreign key*
xid	*X spatial foreign key*
yid	*Y spatial foreign key*
numobs	*Number of observations in the cell*
trajinit	*Number of trajectories starting in the cell*
vmax	*Maximum speed of trajectories in the cell*
distance	*Total distance covered by trajectories in the cell*
time	*Total time spend by the trajectories in the cell*
presence	*Number of trajectories in the cell - distributive*
xborder	*Number of trajectories crossing the x cell border*
yborder	*Number of trajectories crossing the y cell border*
tborder	*Number of trajectories crossing the t cell border*
speed	*Average speed of trajectories in the cell*

Each tuple stored in the *fact table* represents a summarization of the measures that are delimited by the borders of the *cell*. The *base cell* are delimited by the *tid, xid* and *yid* values. The measures presented in the Table 4 are detailed in the Section 2.2. The measures *presence, xborder, yborder* and *tborder* are necessary in order to compute the *holistic presence* measure. These measures are specially important when is necessary to compute the *roll-up* operations, this procedure will be explained in the Section 3.2.

Table 5. X or Y Dimension Table

xid	primary key
xl1	first level of hierarchy
xl2	second level of hierarchy

Table 6. T Dimension Table

tid	primary key
tl1	first level of hierarchy
tl2	second level of hierarchy

3.1. Loader Component

The loader component also is responsible by the settings of the environment in order to receive the data volume. This component loads the data volume into a buffer table (see Table 7). We are considering that this process happens in an unpredictable and unbounded way, therefore we have to store packages of data into a buffer table. This procedure permits to release space in the buffer table, it can be done by the exclusion of the tuples of the trajectories ended.

Table 7. Buffer Table

oid	*Object Identifier*
xvalue	*X spatial value*
yvalue	*Y spatial value*
tvalue	*T time value*
dift	*Time variation between two consecutive positions*
difx	*X spatial variation between two consecutive positions*
dify	*Y spatial variation between two consecutive positions*
dist	*Distance covered between two consecutive positions*
vel	*Speed between two consecutive positions*
idrow	*Identifier of the row*
timestamp	*Timestamp of the observation*

Through the loader component is possible to define the details of the environment and to compute the interpolation procedure. In order to compute the interpolation and to load the *TDW* is necessary to define two very important parameters:

- *Granularity level*
- *Dimension hierarchical level*

The definition of the granularity level is necessary in order to define the regular grid which divides the spatio-temporal environment. This procedure is done before the loading the *TDW*, because in the *TDW* schema we have computed the measures for each cell. Therefore, the definition of the cells is the first step in the loading process. After the definition of the granularity level is possible to define the hierarchy level of the dimension tables, this procedure also can be done by the loader component.

The Figure 6 shows a visualization of the interface available in order to define those settings. These procedures are executed just one time, before the beginning of the loading the data warehouse. After the definition of the settings explained above, the interface permits to start the process of the receiving the data stream values and loading the *TDW*. The Algorithm 1 presents the basic procedures in order to complete the loader process, where C_{cur} represents the *current base cell*, C_{prev} the previous *base cell* stored in the *buffer* related to the same trajectory, and IP represents the set of *base cells* computed by the interpolation process. Using the setting values already defined, the loader

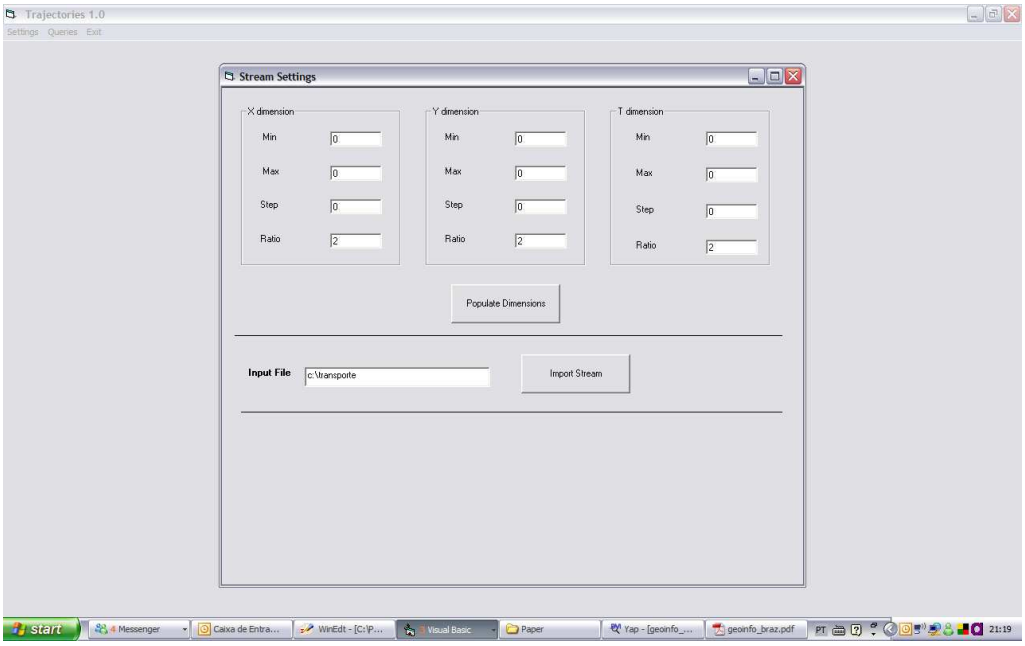

Figure 6. Loading Settings

Algorithm 1 Loading

Input: {*Stream SO of observations - id, x, y, t*}
Output: {*Fact Table FT*}
$FT \longleftarrow \emptyset$
$buffer \longleftarrow \emptyset$
repeat
  $obs \longleftarrow next(SO)$
  $C_{cur} \longleftarrow findCell(obs.x, obs.y, obs.t)$
  if $obs.id \notin buffer$ **then**
    $insertbuffer(obs.id)$
  end if
  $C_{prev} \longleftarrow findCell(obs.x, obs.y, obs.t)$
  $IP \longleftarrow interp(C_{cur}, C_{prev})$
  $ct \longleftarrow 1$
  repeat
    if $IP[ct] \notin FT$ **then**
      $insert(IP[ct], FT)$
    else
      $update(IP[ct], FT)$
    end if
    $ct \longleftarrow ct + 1$
  until $ct < IP.numpoint$
until

component gets the active values in the *buffer table* and performs the procedures in order to load the *TDW*. In the Section 2.1 we have described the concept of *interpolation* used in order to load the *TDW*. The *buffer table* is used in order to implement that procedure. For each *active* row of the *buffer table* the application has to find the related *cell* in the *TDW*. If exists a row in the fact table for that *cell*, the values of *distributive* measures (e.g *numobs, trajinit*) can be updated, else a new row will be inserted in the fact table. It is the procedure when is not necessary to compute the interpolation. However, there are some measures which is necessary to compute the interpolation. In this case the procedure must use the identifier of the active trajectories, their last processed point, the cell such point belongs to, and the speed in the segment ending at such a point. Using that set of values is possible to determine the additional points in order to represent the intersections among the trajectory and the borders of the cells. The additional points can be computed considering the constant speed of the trajectory and the spatio-temporal granularity. In the Algorithm 1, the functions $interp$ is responsible to compute the additional points of the interpolation process. For each new point computed by the interpolation process happens the task of searching the related *cell* in the *TDW*. When there is the base *cell* in the *fact table* the related measures must be updated. Otherwise, the procedure is to insert a tuple with the new values of the measures.

3.2. Aggregation Component

The *aggregation problem* was explained in the Section 2.2. The *distributive* and *algebraic* measures can be solved without problem, but the situation is totaly different when is necessary to compute aggregation measures. In that case there is not a precisely result, just an approximation value can be computed.

In our application the pre-aggregated values are stored in the *TDW*. These pre-aggregated values are computed in the *loading* phase. However, when is necessary to compute some query or to perform a roll-up operation, the application uses the *aggregation component*. The Figure 4 shows the results of a query defined by the boundaries of the cell selected on the map. The results are represented by the charts (right side) where is possible to verify the evolution of the measures for each value of time dimension.

The user can choose the query *base cell* either by the usage of the combo-boxes or by clicking on the map. The map is divided into a regular grid, this division is done in agreement with the granularity defined by the user. In some cases just a simple query in the data warehouse can solve the query, for example when the query is limited in a only one *base cell*, it is possible because the tuples in the *TDW* stores the pre-aggregated values. However, when is necessary to compute a roll-up operation a simple search in the *fact table* is not sufficient. In these cases the solution can not be found by the usage of the relational operators (*Select, Update...*), then the application must use a lot of *stored procedures* developed in order to solve these situations. The computation of the *holistic* measures also is done by the usage of the *stored procedures*. For example, to compute the *holistic presence* measure we need the *distributive* measures *presence* and the other ones *algebraic* measures: *xborder, yborder* and *tborder*. It is a complex task, because is necessary to determine the direction of the *roll-up* operation. If the *roll-up* happens just in the *X-dimension* the only ones measures used will be the *distributive presence* and the other one *algebraic* measure: *xborder*, the same procedure happens for the another dimensions. There is another one more complex query: the *roll-up* in *X, Y* and *T* dimensions, again this

situation is solved by the usage of the *stored procedures*. All the *stored procedures* are invocable by the *loading component* and executed in the *TDW*, the results of the queries are presented in the *Results* area in the query form of the application (see Figure 4).

4. Conclusions and Future Work

In this paper we have presented an application in order to implement a *TDW* considering the constraints to load the data warehouse and compute the aggregation values. The application is a first step in order to try solving the problems of loading and aggregation in a *TDW* environment. The data cube model adopted is very simple, it is just a contribution in order to improve the discussion about the problems to implement a *TDW* model. However, this model can be extended to more general situations.

The current stage of the application can solve some problems of a trajectory data warehouse environment. However, the dimensions that we have used are very simple, a possible future work could be to sophisticate the hierarchy of the dimensions. The loading phase is an opened problem, we have limited the loading phase considering a linear interpolation, but is possible to find some topological situations (e.g roads, bridges) where is very difficult to do this interpolation because of the some constraints in the movement of the object.

In this work we have used the *roll-up* operation, however could be interesting to offer mechanisms in order to compute other operators such as drill-down, pivot, slice and dice. Therefore, the development of a query language using OLAP operators also is a possible point to research.

Another interesting area of additional research is to develop another more complex measures in order to provide values to discover patterns or trend of the trajectories. To compute values to support the data mining tasks is a very interesting point of future research.

References

Braz, F., Orlando, S., Orsini, R., Raffaetà, A., Roncato, A., and Silvestri, C. (2007). Approximate aggregations in trajectory data warehouse. In *Proc. of ICDE Workshop STDM*, pages 536–545.

Brinkhoff, T. (2000). Generating network-based moving objects. In *SSDBM '00: Proceedings of the 12th international Conference on Scientific and Statistical Database Management*, page 253, Washington,DC,USA. IEEE Computer Society.

Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., and Pirahesh, H. (1997). Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–54.

Han, J., Stefanovic, N., and Kopersky, K. (1998). Selective materialization: An efficient method for spatial data cube construction. *PAKDD'98*, pages 144–158.

Kimball, R. (1996). *Data Warehouse Toolkit*. John Wiley.

Marchant, P., Briseboi, A., Bedard, Y., and Edwards, G. (2004). Implementation and evaluation of a hypercube-based method for spatiotemporal exploration and analysis. *ISPRS Journal of Photogrammetry and Remote Sensing*, 59:6–20.

Rivest, S., Bedard, Y., and Marchand., P. (2001). Towards better support for spatial decision making: Defining the characteristics of spatial on-line analytical processing(solap). *Geomatica*, 55(4):539–555.

Shekhar, S., Lu, C., Tan, X., Chawla, S., and Vatsavai, R. (2001). *Map Cube: Avisualization Tool for Spatial Data Warehouse*, chapter Geographic Data Mining and Knowledge Discovery. Taylor and Francis.