Building Real-World Trajectory Warehouses

Gerasimos Marketos
Dept. of Informatics,
University of Piraeus, Greece
marketos@unipi.gr

Elias Frentzos
Dept. of Informatics,
University of Piraeus, Greece
efrentzo@unipi.gr

Irene Ntoutsi
Dept. of Informatics,
University of Piraeus, Greece
ntoutsi@unipi.gr

Nikos Pelekis
Dept. of Informatics,
University of Piraeus, Greece
npelekis@unipi.gr

Alessandra Raffaetà
Dept. of Informatics,
University Ca' Foscari Venezia, Italy
raffaeta@dsi.unive.it

Yannis Theodoridis
Dept. of Informatics,
University of Piraeus, Greece
ytheod@unipi.gr

ABSTRACT

The flow of data generated from low-cost modern sensing technologies and wireless telecommunication devices enables novel research fields related to the management of this new kind of data and the implementation of appropriate analytics for knowledge extraction. In this work, we investigate how the traditional data cube model is adapted to trajectory warehouses in order to transform raw location data into valuable information. In particular, we focus our research on three issues that are critical to trajectory data warehousing: (a) the trajectory reconstruction procedure that takes place when loading a moving object database with sampled location data originated e.g. from GPS recordings, (b) the ETL procedure that feeds a trajectory data warehouse, and (c) the aggregation of cube measures for OLAP purposes. We provide design solutions for all these issues and we test their applicability and efficiency in real world settings.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications

General Terms: Design

Keywords: Trajectory Reconstruction, Moving Object Databases, Trajectory Data Warehouses, Distinct Count problem

1. INTRODUCTION

The usage of location aware devices, such as mobile phones and GPS-enabled devices, is widely spread nowadays, allowing access to large spatiotemporal datasets. The space-time nature of this kind of data results in the generation of huge amounts of trajectory data and imposes new challenges regarding their efficient management. To address this need, the traditional database technology has been extended into Moving Object Databases (MODs) that handle modeling, indexing and query processing issues for trajectories [6], [16]. Moreover, the analysis of such trajectory data raises opportunities for discovering behavioral patterns that can be exploited in applications like traffic management and service accessibility. Online analytical processing (OLAP) and data mining (DM) techniques have been employed in order to convert this vast amount of raw data into useful knowledge [8], [9], [12]. Indicatively, the variable number of moving objects in different urban areas, the average speed of vehicles, the ups and downs of vehicles' speed as well as useful insights, like discovering popular movements [4] can be analyzed in a Trajectory Data Warehouse (TDW).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MobiDE'08, June 13, 2008, Vancouver, British Columbia, Canada.
Copyright 2008 ACM 978-1-60558-221-4

In this paper, we propose a framework for TDW that takes into consideration the complete flow of tasks required during a TDW development. The complete lifecycle of a TDW is illustrated in Figure 1 and it consists of various steps. A *Trajectory Reconstruction process* is applied on the raw time-stamped location data in order to generate trajectories, which are then stored into a *MOD*. Then, an *Extract-Transform-Load* (*ETL) procedure* is activated that feeds the data cube(s) with aggregate information on trajectories. The final step of the process offers OLAP (and, eventually, DM) capabilities over the aggregated information contained in the trajectory cube model.

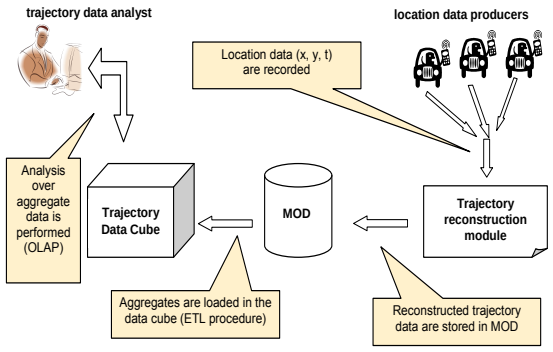

Figure 1. The architecture of our framework.

A MOD maintains object locations recorded at various time points in the form of trajectories. Formally, let $D = \{T_1, T_2, \ldots, T_N\}$ be a MOD consisting of the trajectories of a set of moving objects. Assuming linear interpolation between consecutive sampled locations, the trajectory $T_i = <(x_{i_1}, y_{i_1}, t_{i_1}), \ldots, (x_{i_{n_i}}, y_{i_{n_i}}, t_{i_{n_i}})>$ consists of a sequence of n_i line segments in 3D space, where each segment represents the continuous "development" of the corresponding moving object between consecutive locations (x_{i_j}, y_{i_j}) sampled at time t_{i_j}. Projecting T_i on the spatial 2D plane (temporal 1D line), we get the *route* r_i (the *lifespan* l_i,

respectively) of moving object. Additional motion parameters can be derived, including the traversed length *len* of *route* r_i, average speed, acceleration, etc.

Let us assume a MOD that stores raw locations of moving objects (e.g. humans); a typical schema, to be considered as a minimum requirement, for such a MOD is illustrated in Figure 2.

OBJECTS (id: *identifier*, description: *text*, gender: {M | F}, birth-date: *date*, profession: *text*, device-type: *text*)

RAW_LOCATIONS (object-id: *identifier*, timestamp: *datetime*, eastings-x: *numeric*, northings-y: *numeric*, altitude-z: *numeric*)

MOD_TRAJECTORIES (trajectory-id: *identifier*, object-id: *identifier*, trajectory: *3D geometry*)

Figure 2. An example of a MOD.

OBJECTS includes a unique object identifier (id), demographic information (e.g. description, gender, date of birth, profession) as well as device-related technographic information (e.g. GPS type). *RAW_LOCATIONS* stores object locations at various time stamps (i.e., samples), while *MOD_TRAJECTORIES* maintains the trajectories of the objects, after the application of the trajectory reconstruction process.

Following the multidimensional model [1], a data cube for trajectories consists of a fact table containing keys to dimension tables and a number of appropriate measures. Dimension tables might have several attributes in order to build multiple hierarchies so as to support OLAP analysis whereas measures could be trajectory-oriented (e.g., number of trajectories, number of objects, average speed, etc.). For each dimension we define a finest level of granularity which refers to the detail of the data stored in the fact table.

Definitely, a TDW should include a *spatial* and a *temporal dimension* describing geography and time, respectively. Another dimension regarding *conventional* information about moving objects (including demographical information, such as gender, age, etc.) could be considered as well.

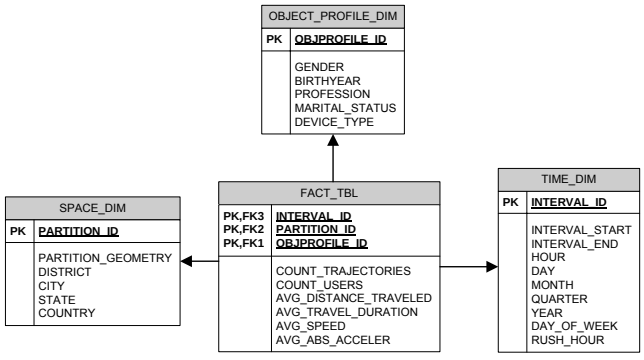

Figure 3. An example of TDW.

Based on the above, we consider as a minimum requirement for our framework the following dimensions (Figure 3):

- *Geography*: the spatial dimension (SPACE_DIM) allows us to define spatial hierarchies. Handling geography at the finest level of granularity could include (as alternative solutions) a simple grid, a road network or even coverage of the space with respect to the mobile cell network. According to the first alternative, the space is divided in explicitly defined (usually, rectangular) areas. For the purposes of this paper, we assume a grid of equally sized rectangles (PARTITION_GEOMETRY in Figure 3), the size of which is a user-defined parameter, (e.g. 10×10 Km^2).
- *Time*: the temporal dimension (TIME_DIM) defines temporal hierarchies. Time dimension has been extensively studied in the data warehousing literature [1]. At the finest level of granularity, we assume user-defined time intervals (e.g. 1 hour periods).
- *User Profile*: the thematic dimension (OBJECT_PROFILE_DIM) refers to demographic and technographic information.

Apart from keys to dimension tables, the fact table also contains a set of measures including aggregate information. The measures considered in the TDW schema of Figure 3 include the *number of distinct trajectories* (COUNT_TRAJECTORIES), the *number of distinct users* (COUNT_USERS), the *average traveled distance* (AVG_DISTANCE_TRAVELED), the *average travel duration* (AVG_TRAVEL_DURATION), the *average speed* (AVG_SPEED) and the *average acceleration* in absolute values (AVG_ABS_ACCELER), for a particular group of people moving in a specific spatial area during a specific time period.

In order to build a TDW, several issues should be handled; we summarize these issues below accompanied with our contributions in this paper:

- First, sampled positions received by GPS-enabled devices need to be converted into trajectory data and to be stored in a MOD; to this end, we propose a *trajectory reconstruction* technique that transforms sequences of raw sample points into meaningful trajectories.
- Second, the TDW is to be fed with aggregate trajectory data; to achieve it we propose two alternative solutions: a (index-based) *cell-oriented* and a (non-index-based) *trajectory-oriented* ETL process.
- Third, aggregation capabilities over measures should be offered for OLAP purposes (i.e., how the measures at a lower level of the cube hierarchy can be exploited in order to compute the measures at some higher level of the hierarchy). The peculiarity with trajectory data is that a trajectory might span multiple base cells (the so called *distinct count problem* [17]). This causes *aggregation* hindrances in OLAP operations. We provide approximation solutions for this problem, which turn out to perform effectively.

The rest of the paper is organized as follows: Section 2 presents basic concepts on trajectories and trajectory warehouses as well as the related work that motivated our work. Section 3 constitutes the core of the paper, where we discuss the trajectory reconstruction process, the ETL procedure for feeding the data cube, and the measures aggregation problem. Empirical results are presented in Section 4, where we evaluate the efficiency of our approach through an extensive experimental study. Conclusions and open research issues are outlined in Section 5.

2. RELATED WORK

The pioneering work by Han et al. [7] introduces the concept of spatial data warehousing (SDW). The authors extend the idea of cube dimensions so as to include spatial and non-spatial ones, and

of cube measures so as to represent space regions and/or calculate numerical data. One step further from modeling a SDW is modeling a TDW. The motivation here is to transform raw trajectories to valuable information that can be utilized for decision making purposes in ubiquitous applications, such as mobile marketing, location-based services and traffic control management. Trajectory warehousing [13] is in its infancy but we can distinguish three major research directions on this field: modeling, aggregation and indexing.

From a modeling perspective, the definition of hierarchies in the spatial dimension introduces issues that should be addressed. The spatial dimension may include not explicitly defined hierarchies [8]. Thus, multiple aggregation paths are possible and they should be taken into consideration during OLAP operations. Tao and Papadias [16] propose the integration of spatial and temporal dimensions and present appropriate data structures that integrate spatiotemporal indexing with pre-aggregation. Choi et al. [2] try to overcome the limitations of multi-tree structures by introducing a new index structure that combines the benefits of Quadtrees and Grid files. However, the above frameworks focus on calculating simple measures (e.g. count customers). Very recently, an attempt to model and maintain a TDW is presented in [10], [11] where a simple data cube consisting of spatial / temporal dimensions and numeric measures concerning trajectories, is defined.

The distinguishing features of our work are:

i) the presence of a preprocessing phase dealing with the explicit construction of the trajectories, which are then stored into a MOD that offers powerful and efficient operations for the manipulation of such data;

ii) the proposal of alternative ETL processes, a procedure underestimated so far in related work; and

iii) the solutions proposed on the challenging issue of measure aggregation which occurs due to the trajectory oriented cube model.

We emphasize that this work does not aim at proposing yet another TDW model. Instead, we provide efficient solutions to support the complete flow of processes in a TDW, from trajectory reconstruction to trajectory-oriented OLAP.

3. PROPOSED SOLUTIONS

So far, we have described a MOD that stores trajectory data reconstructed from raw location data (Figure 2) and a trajectory-oriented data cube that offers multi-dimensional analysis capabilities (Figure 3). In this section, we thoroughly describe our proposed solutions for the trajectory reconstruction problem (Subsection 3.1), the efficient ETL process for feeding the TDW (Subsection 3.2) and the distinct count problem that appears during measures aggregation (Subsection 3.3).

3.1 Reconstructing trajectories

As already discussed, collected raw data represent time-stamped geographical locations (Figure 4a). Apart from storing these raw data in the MOD, we are also interested in reconstructing trajectories (Figure 4b). The so-called *trajectory reconstruction* task is not a straightforward procedure. Having in mind that raw points arrive in bulk sets, we need a filter that decides if the new series of data is to be *appended* to an existing trajectory or not.

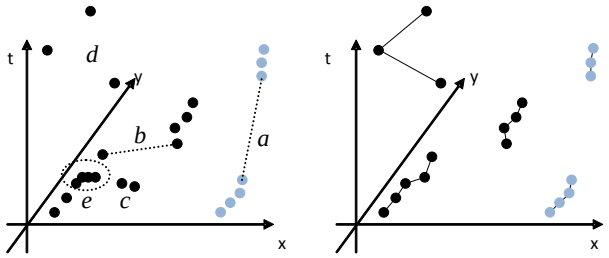

Figure 4. a) raw locations, b) reconstructed trajectories.

In this work, we assume this filter to be part of a trajectory reconstruction manager, along with a simple method for determining different trajectories, which applies it on raw positions. Due to the fact that the notion of trajectory cannot be the same in every application, we define the following generic trajectory reconstruction parameters:

- *Temporal gap between trajectories* gap_{time}: the maximum allowed time interval between two consecutive time-stamped positions of the same trajectory for a single moving object. As such, any time-stamped position of object o_i, received after more than gap_{time} time units from its last recorded position, will cause a new trajectory of the same object to be created (case *a* in Figure 4a) .
- *Spatial gap between trajectories* gap_{space}: the maximum allowed distance in 2D plane between two consecutive time-stamped positions of the same trajectory. As such, any time-stamped position of object o_i, with distance from the last recorded position of this object greater than gap_{space}, will cause a new trajectory to be created for o_i (case *b* in Figure 4a).
- *Maximum speed* V_{max}: the maximum allowed speed of a moving object. It is used in order to determine whether a reported time-stamped position must be considered as noise and consequently discarded from the output trajectory. When a new time-stamped location of object o_i is received, it is checked with respect to the last known position of that object, and the corresponding instant speed is calculated. If it exceeds V_{max}, this location is considered as noise and (temporarily) it is not considered in the trajectory reconstruction process (however, it is kept separately as it may turn out to be useful again – see the parameter that follows) (case *c* in Figure 4a).
- *Maximum noise duration* $noise_{max}$: the maximum duration of a noisy part of a trajectory. Any sequence of noisy time-stamped positions of the same object will result in a new trajectory given that its duration exceeds $noise_{max}$. For example, consider an application recording positions of pedestrians where the maximum speed set for a pedestrian is V_{max} = 3 m/sec. When he/she picks up a transportation mean (e.g., a bus), the recorded instant speed will exceed V_{max}, flagging the positions on the bus as noise. The maximum noise length parameter stands for supporting this scenario: when the duration of this sequence of 'noise' exceeds $noise_{max}$, a new trajectory containing all these positions is created (case *d* in Figure 4a).
- *Tolerance distance* D_{tol}: the tolerance of the transmitted time-stamped positions. In other words, it is the maximum distance between two consecutive time-stamped positions of the same object in order for the object to be considered as stationary. When a new time-stamped location of object o_i is received, it is

checked with respect to the last known position of that object, and if the distance of the two locations is smaller than D_{tol}, it is considered redundant and consequently discarded (case *e* in Figure 4a).

The proposed TRAJECTORY-RECONSTRUCTION algorithm is illustrated in Figure 5. The input of the algorithm includes raw data points (i.e., time-stamped positions) along with object-id, and a list containing the partial trajectories processed so far by the *trajectory reconstruction manager*; these partial trajectories are composed by several of the most recent trajectory points, depending on the values of the algorithm parameters.

```
Algorithm Trajectory-Reconstruction
        (PartialTrajectories List, P Point, OId ObjectId)

 1.    IF NOT PartialTrajectories.Contains(OId) THEN
 2.      CTrajectory=New Trajectory;
 3.      CTrajectory.AddPoint(P);
 4.      PartialTrajectories.Add(CTrajectory);
 5.    ELSE
 6.      CTrajectory=PartialTrajectories(OId);
 7.      IF Distance(CTrajectory.LastPoint,P)<= DTOL THEN
 8.        IF P.T - CTrajectory.LastPoint.T > gapTime THEN
 9.          Report Ctrajectory.LastPoint;
10.          CTrajectory.Id=CTrajectory.Id+1;
11.          CTrajectory.AddPoint(P);
12.        ENDIF
13.      ELSEIF Speed(CTrajectory.LastPoint,P)> Vmax THEN
14.        IF P.T - CTrajectory.LastPoint.T > noisemax THEN
15.          Report CTrajectory.Noise;
16.        ELSE
17.          CTrajectory.AddNoise(P);
18.        ENDIF
19.      ELSEIF Distance(CTrajectory.LastPoint,P)> gapspace THEN
20.        Report Ctrajectory.LastPoint;
21.        CTrajectory.Id=CTrajectory.Id+1;
22.        CTrajectory.AddPoint(P);
23.      ELSE
24.        IF P.T - CTrajectory.LastPoint.T > gapTime THEN
25.          Report Ctrajectory.LastPoint;
26.          CTrajectory.Id=CTrajectory.Id+1;
27.          CTrajectory.AddPoint(P);
28.        ELSE
29.          CTrajectory.AddPoint(P);
30.        ENDIF
31.      ENDIF
32.    ENDIF
```

Figure 5. The TRAJECTORY-RECONSTRUCTION algorithm.

As a first step (lines 1-6), the algorithm checks whether the object has been processed so far, and, if so, retrieves its partial trajectory from the corresponding list, while, in the opposite case, creates a new trajectory and adds it to the list. Then (lines 7-31), it compares the incoming point P with the tail of the partial trajectory (LastPoint) by applying the above mentioned trajectory reconstruction parameters:

- it rejects P if it is closer than D_{tol} to LastPoint (lines 7-12) or
- it rejects P when a speed greater than V_{max} is calculated, unless the $noise_{max}$ case is triggered (lines 13-18) or
- it creates a new trajectory if the temporal duration between P and LastPoint is longer than gap_{time} (lines 8-12 and 24-27) or their spatial distance is greater than gap_{space} (lines 19-22);

in any other case, it reports LastPoint in the partial trajectory and replaces it with P.

The above procedure supports the reasonable requirement for detecting one trajectory per trip: Let us consider the case where a tracked user is traveling from home to work in the morning and from work to home in the evening, leaving his/her tracking device (e.g., GPS) always active. In this case, during the time the car is parked there are no spatial gaps, and no maximum speed problems which may cause a new trajectory creation. Moreover, the GPS outputs a position every second, so no temporal gaps initially exist; however, since the car is not moving, the algorithm eliminates all points reported during the non-moving interval, and, an artificial temporal gap is created (i.e., only the first point after the car parking and the last before starting moving again exist in the trajectory reconstruction algorithm). As a consequence, the algorithm detects the temporal gap and creates new trajectories, as needed, based only on the information that the tracked object stopped moving for a sufficiently large temporal period (i.e., greater than gap_{time}).

3.2 ETL processing over trajectory data

Once trajectories have been constructed and stored in a MOD, the ETL phase is executed in order to feed the TDW. Loading data into the dimension tables is straightforward; however, this is far more complex for the fact table. In particular, recalling Figure 3, the main task is to fill in the measures with the appropriate numeric values for each of the base cells that are identified by the three foreign keys (PARTITION_ID, INTERVAL_ID, OBJPROFILE_ID) of the fact table.

The COUNT_TRAJECTORIES measure for a base cell *bc* is calculated by counting all the distinct trajectory ids that pass through *bc*. The COUNT_USERS measure for a base cell *bc* is calculated similarly by counting all the distinct object ids that pass through *bc*.

In order to calculate the AVG_DISTANCE_TRAVELED measure for a base cell *bc* we define an *auxiliary* measure, called SUM_DISTANCE as the summation of the length *len(TP)* of each portion *TP* of the trajectories lying within *bc*. More formally,

$$SUM\_DISTANCE(bc) = \sum_{TP_i \in bc} len(TP_i)$$

Then, the AVG_DISTANCE_TRAVELED measure is computed by dividing the SUM_DISTANCE by the COUNT_TRAJECTORIES measure:

$$AVG\_DISTANCE\_TRAVELED(bc) = \frac{SUM\_DISTANCE(bc)}{COUNT\_TRAJECTORIES(bc)}$$

Similar is the case for the AVG_TRAVEL_DURATION measure:

$$AVG\_TRAVEL\_DURATION(bc) = \frac{SUM\_DURATION(bc)}{COUNT\_TRAJECTORIES(bc)}$$

where, SUM_DURATION is also an auxiliary measure defined as the summation of the duration *lifespan(TP)* of each portion *TP* of the trajectories inside *bc*.

$$SUM\_DURATION(bc) = \sum_{TP_i \in bc} lifespan(TP_i)$$

In the same fashion, the AVG_SPEED measure is calculated by dividing the auxiliary measure SUM_SPEED (i.e. the sum of the speeds of each portion *TP* inside *bc*) with COUNT_TRAJECTORIES:

$$AVG\_SPEED(bc) = \frac{SUM\_SPEED(bc)}{COUNT\_TRAJECTORIES(bc)}$$

where $SUM\_SPEED(bc) = \sum_{TP_i \in bc} \frac{len(TP_i)}{lifespan(TP_i)}$

Likewise, the AVG_ABS_ACCELER is a suchlike fraction

$$AVG\_ABS\_ACCELER(bc) = \frac{SUM\_ABS\_ACCELER(bc)}{COUNT\_TRAJECTORIES(bc)}$$

where SUM_ABS_ACCELER is a supplementary measure that summates the absolute accelerations of all portions *TP* lying in *bc*

$$SUM\_ABS\_ACCELER(bc) = \sum_{TP_i \in bc} \frac{\left| speed_{fin}(TP_i) - speed_{init}(TP_i) \right|}{lifespan(TP_i)}$$

and $speed_{fin}$ ($speed_{init}$) is the finally (initially, respectively) recorded speed of the trajectory portion (TP_i) in *bc*.

It is important to remark that all these measures are computed in an exact way by using the MOD. In fact our MOD Hermes [14] provides a rich palette of spatial and temporal operators for handling trajectories. Unfortunately, rolling-up these measures is not straightforward due to the count distinct problem [17] as it will be discussed in detail in the next subsection.

As already mentioned, in order to calculate the measures of the data cube, we have to extract the portions of the trajectories that fit into the base cells of the cube. We consider a MOD of *U* user profiles, *N* trajectories, *M* spatial partitions and *K* temporal intervals. We propose two alternative solutions to this problem: (i) a cell-oriented and (ii) a trajectory-oriented approach.

According to the *cell-oriented approach* (COA), we search for the trajectory portions that lie within the base cells. The ETL procedure for feeding the fact table of the TDW is described by the proposed CELL-ORIENTED-ETL algorithm (Figure 6). First, we search for the portions of trajectories under the concurrent constraint that they reside inside a spatiotemporal cell *C* (line 4). Then, the algorithm proceeds to the decomposition of the portions with respect to the user profiles they belong to (lines 6-9).

```
Algorithm Cell-Oriented-ETL(D MODTrajectoryTable)
 1. // For each pair <Region, Interval> forming a s-t cell Cj
 2. FOR EACH cell Cj DO
 3.    // Find the set of sub-trajectories inside the cell
 4.    S = intersects(D, Cj);
 5.    // Decompose S to subsets according to object profile
 6.    FOR EACH subset S' of S DO
 7.        // Compute the various measures
 8.        Compute_Measures(S');
 9.    END-FOR
10. END-FOR
```

Figure 6. The CELL-ORIENTED-ETL algorithm.

The efficiency of the above described *COA* solution depends on the effective computation of the parts of the moving object trajectories that reside in the spatiotemporal cells (line 4). This step is actually a spatiotemporal range query that returns not only the identifiers but also the portions of trajectories that satisfy the range constraints. To efficiently support this trajectory-based query processing requirement, we employ the TB-tree [15], a state-of-the-art index for trajectories that can efficiently support trajectory query processing.

According to the *trajectory-oriented approach* (TOA), we discover the spatiotemporal cells where each trajectory resides in (line 6). In order to avoid checking all cells, we use (line 4) a rough approximation of the trajectory, its Minimum Bounding Rectangle (MBR), and we exploit the fact that the granularity of cells is fixed in order to detect (possibly) involved cells in constant time. Then, we identify the portions of the trajectory that fits into each of those cells (lines 8-15). This ETL procedure is described by the proposed TRAJECTORY-ORIENTED-ETL algorithm (Figure 7).

```
Algorithm Trajectory-Oriented-ETL(D MODTrajectoryTable)
 1. // For each Trajectory Ti
 2. FOR EACH Trajectory Ti of D DO
 3.    // Find the Minimum Bounding Rectangle of Ti
 4.    MBRTi = Compute_MBR(Ti);
 5.    // Find the set of s-t cells C that overlap with the MBR
 6.    O = Overlap(C, MBRTi)
 7.    // Find the portions (P) of trajectory Ti inside each cell
 8.     FOR EACH O' of O DO
 9.      P = singlet_intersects(Ti, O');
10.      //If the cell contains portions of the trajectory
11.      IF(P NOT NULL) THEN
12.          // Compute the various measures
13.          Compute_Measures(P);
14.      END-IF
15.     END-FOR
16. END-FOR
```

Figure 7. The TRAJECTORY-ORIENTED-ETL algorithm.

3.3 Addressing the distinct count problem

During the ETL process, measures can be computed in an accurate way by executing MOD queries based on the formulas provided in the previous section. However, once the fact table has been fed, the trajectory and user identifiers are not maintained and only aggregate information is stored inside the TDW.

The aggregate functions computing the super-aggregates of the measures are categorized by Gray et al. [5] into three classes based on the complexity required for this computation, starting from a set of already available sub-aggregates:

- *distributive* (the super-aggregates can be computed from the sub-aggregates),
- *algebraic* (the super-aggregates can be computed from the sub-aggregates with a finite set of auxiliary measures), and
- *holistic* (the super-aggregates cannot be computed from sub-aggregates, even if we employ auxiliary measures).

In our case, the aggregate functions to obtain super-aggregates for the main measures discussed in Subsection 3.2 are classified as holistic and as such they require the MOD data to compute super-aggregates in all levels of dimensions. This is due to the fact that COUNT_USERS, COUNT_TRAJECTORIES and, as a consequence, the other measures defined in terms of COUNT_TRAJECTORIES are subject to the distinct count problem [17]: if an object remains in the query region for several timestamps during the query interval, instead of counting this object once, it is counted multiple times in the result.

Notice that once a technique for rolling-up the COUNT_TRAJECTORIES measure is devised, it is straightforward to define a roll-up operation for the AVG measures. In fact the latter can be implemented as the sum of the corresponding auxiliary measures divided by the result of the roll-up of COUNT_TRAJECTORIES. As such, diminishing the calculations in the numerator, hereafter, we focus on the (denominator) number of distinct trajectories (COUNT_TRAJECTORIES); COUNT_USERS is handled in a similar way.

In order to implement a roll-up operation over this measure, a first solution is to define a distributive aggregate function which simply obtains the super-aggregate of a cell *C* by summing up the measures COUNT_TRAJECTORIES in the base cells composing *C*. In the literature, this is a common approach to aggregate spatio-

temporal data but, as we will show in Section 4, it produces a very rough approximation. Following the proposal in [10], an alternative solution is to define an algebraic aggregate function. The idea is to store in the base cells a tuple of auxiliary measures that will help us to correct the errors caused due to the duplicates when rolling-up.

More formally, let $C_{(x,y),t,p}$ be a base cell, which contains, among the others, the following measures (it is worth noting that these measures are loaded without errors into the base cells, by exploiting the MOD functionalities):

- $C_{(x,y),t,p}$.COUNT_TRAJECTORIES: the number of distinct trajectories of profile *p* intersecting the cell ($C_{(x,y),t,p}$.*Traj* for short).
- $C_{(x,y),t,p}$.*cross-x*: the number of distinct trajectories of profile *p* crossing the *spatial* border between $C_{(x-1,y),t,p}$ and $C_{(x,y),t,p}$, where $C_{(x-1,y),t,p}$ is the adjacent cell (on the left) along with *x*-axis.
- $C_{(x,y),t,p}$.*cross-y*: the number of distinct trajectories of profile *p* crossing the *spatial* border between $C_{(x,y-1),t,p}$ and $C_{(x,y),t,p}$, where $C_{(x,y-1),t,p}$ is the adjacent cell (below) along with *y*- axis.
- $C_{(x,y),t,p}$.*cross-t*: the number of distinct trajectories of profile p crossing the *temporal* border between $C_{(x,y),t-1,p}$ and $C_{(x,y),t,p}$, where $C_{(x,y),t-1,p}$ is the adjacent cell (below) along with *t*- axis.

Let $C_{(x',y'),t',p'}$ be a cell consisting of the union of two adjacent cells with respect to a spatial/temporal dimension, for example $C_{(x',y'),t',p'} = C_{(x,y),t,p} \cup C_{(x+1,y),t,p}$ (when aggregating along *x*- axis). In order to compute the super-aggregate corresponding to $C_{(x',y'),t',p'}$, we proceed as follows:

$$C_{(x',y'),t',p'}.Traj = C_{(x,y),t,p}.Traj + C_{(x+1,y),t,p}.Traj - C_{(x+1,y),t,p}.cross\text{-}x$$

The other measures associated with $C_{(x',y'),t',p'}$ can be computed as follows:

$$C_{(x',y'),t',p'}.cross\text{-}x = C_{(x,y),t,p}.cross\text{-}x$$

$$C_{(x',y'),t',p'}.cross\text{-}y = C_{(x,y),t,p}.cross\text{-}y + C_{(x+1,y),t,p}.cross\text{-}y$$

$$C_{(x',y'),t',p'}.cross\text{-}t = C_{(x,y),t,p}.cross\text{-}t + C_{(x+1,y),t,p}.cross\text{-}t$$

The computation of $C_{(x',y'),t',p'}.Traj$ can be thought of as an application of the well-known Inclusion/Exclusion principle for sets: $|A \cup B| = |A| + |B| - |A \cap B|$. Note that in some cases $C_{(x+1,y),t,p}.cross\text{-}x$ is not equal to $|A \cap B|$, and this may introduce errors in the values returned by this algebraic function. In fact, if a trajectory is fast and agile, it can be found in both $C_{(x,y),t,p}$ and $C_{(x+1,y),t,p}$ without crossing the *X* border (since it can reach $C_{(x+1,y),t,p}$ by crossing the *Y* borders of $C_{(x,y),t,p}$ and $C_{(x+1,y),t,p}$).

It is worth noticing that the agility of a trajectory affects the error in the roll-up computation. In fact, a trajectory coming back to an already visited cell can produce an error. In the following figures we illustrate the two main kinds of error that the algebraic aggregate function can introduce.

In Figure 8a, if we group together the cells C_3 and C_4, we obtain that the number of distinct trajectories is $C_3.Traj + C_4.Traj - C_4.cross\text{-}x = 1+1-0 = 2$. This is an *overestimate* of the number of distinct trajectories. On the other hand, in Figure 8b, if we group together C_1 and C_2 we correctly obtain $C_1.Traj + C_2.Traj - C_2.cross\text{-}x = 1+1-1 = 1$, similarly by aggregating C_3 and C_4. However, if we group $C_1 \cup C_2$ with $C_3 \cup C_4$ we obtain $C_1 \cup C_2.Traj + C_3 \cup C_4.Traj - C_1 \cup C_2.cross\text{-}y = 1+1-2 = 0$. This is an *underestimate* of the number of distinct trajectories.

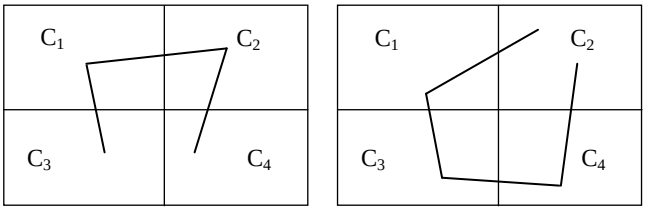

Figure 8. a) Overestimate of *Traj.* b) Underestimate of *Traj.*

In order to give a bound to this kind of errors, let us focus on a single trajectory. This is not a limitation because the values of the measures *Traj*, *cross-x*, *cross-y*, and *cross-t* can be computed by summing up the contributions given to such a measure by each trajectory in isolation. Since the aggregation operations are linear functions, the above property also holds for aggregated cells.

First of all, let us introduce the concept of *uni-octant sequence*. We call *uni-octant sequence* a *maximal* sequence of connected segments of a trajectory whose slopes are in the same octant. It is evident that a trajectory can be *uniquely* decomposed into uni-octant sequences.

A uni-octant sequence *us* can traverse a cell *C* only once, i.e. if *us* starts from *C* it can only exit from *C*, otherwise it can only enter once in *C*. As a consequence, if a trajectory consists of a single uni-octant sequence it does not produce any error in the roll-up computation for the measure COUNT_TRAJECTORIES. In fact, as discussed above, errors can only arise when a trajectory visits a cell at least twice.

This can be generalised to a trajectory *T* composed by several uni-octant sequences. In this case, the computed value of the measure *Traj* in an *aggregated* cell *C* is limited by the number of uni-octant sequences of *T* intersecting *C*. This is an upper bound that can be reached, as shown in Figure 8a.

Note that in order to face the distinct count problem when aggregating cells with different profiles, analogously to what we did for the spatial and temporal dimensions, it could be helpful to consider a measure *cross-P*, specifying the number of distinct trajectories changing their profile from one cell to an adjacent one. However, since profile changes are rather rare in real-world scenarios and only appear in long term situations, we omit computing *cross-P* and we simply use the distributive aggregate function *sum* for this kind of aggregations. (In any case, when there is need *cross-P* can be added in our framework without additional difficulty.)

4. EXPERIMENTATION WITH A REAL-WORLD TDW

In this section, we evaluate the proposed solutions by implementing the TDW architecture (Figure 3) for a real-world application. More specifically, we used a large real dataset: a part of the e-Courier dataset [3] consisting of 6.67 millions of raw location records (a file of 504 Mb, in total), that represent the movement of 84 couriers moving in London during a one month period (July 2007) with a 10 sec sample rate. For all the experiments we used a PC with 1 Gb RAM and P4 3 GHz CPU.

Default values of the trajectory reconstruction parameters were set as follows: temporal gap 900 sec, spatial gap 5 Km, maximum speed 50 m/s, maximum noise duration 600 sec, and tolerance distance 20 m, and the volume of the raw locations dataset varied from 0.5 millions of records up to the full size available. This setting resulted in a maximum of 4263 trajectories (which corresponds to an average 1.64 trajectories per courier per day).

We concluded in the above values of parameters after analyzing the behaviour of different candidate values and assessing the number of produced trajectories. As a first consideration, the selected value for maximum speed of vehicles is reasonable. Secondly, different values of distance accuracy parameter have not an effect on the number of produced trajectories. Below, we illustrate the results of analysis for: temporal gap (Figure 9), maximum noise duration (Figure 9) and the spatial gap parameters (Figure 10). The number of produced trajectories seems to stay stable from the point selected on.

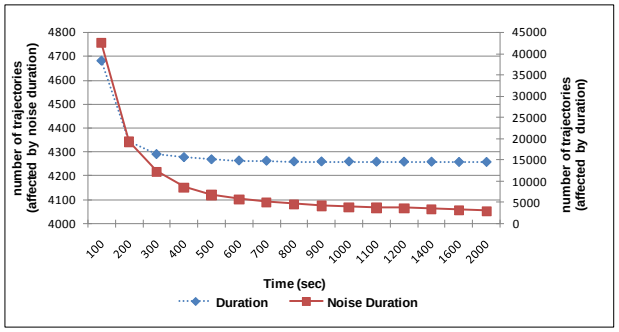

Figure 9. The effect of temporal gap and noise duration parameters

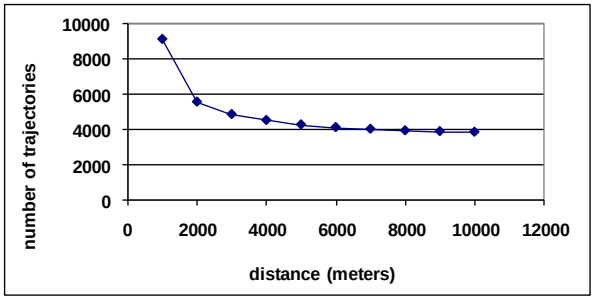

Figure 10. The effect of spatial gap parameter

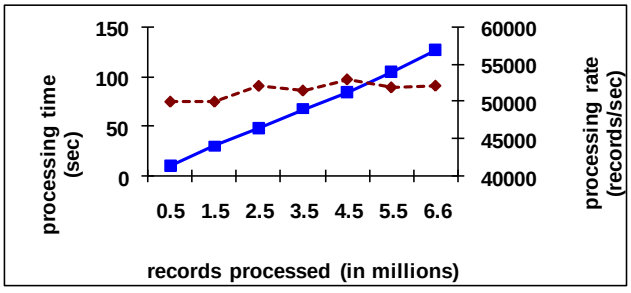

Figure 11. Performance of trajectory reconstruction (solid line: processing time; dotted line: processing rate)

The next experiment, illustrated in Figure 11, is about the efficiency of the TRAJECTORY-RECONSTRUCTION algorithm, proposed in Subsection 3.1. It is clear that the TRAJECTORY-RECONSTRUCTION algorithm performs linear with the size of the input dataset (and allows the processing of the full dataset in about 2 min). Furthermore, the average processing rate is almost stable (~ 50K records/sec).

In the next set of experiments, we evaluate the effectiveness of the TRAJECTORY-RECONSTRUCTION algorithm in the case of a large number of users and towards the goal of processing input in real-time. In particular, we measure its processing time for various sample rates, as illustrated in Figure 12. According to this experiment, by choosing e.g. a 20 sec sample rate the algorithm can handle in real time up to 1000K objects, which is a really large number for real-world applications (at least nowadays). The conclusion from this experiment is that the proposed TRAJECTORY-RECONSTRUCTION algorithm is effective for real-time processing by keeping a trade-off between the number of users to be supported and the sample rate set for transmitting their location.

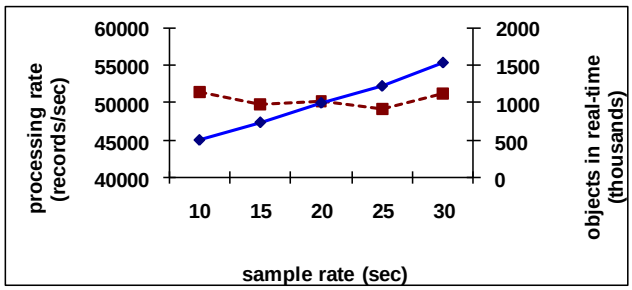

Figure 12. The effect of sample rate in real-time processing (solid line: objects in real-time; dotted line: processing rate)

For the evaluation of the ETL process we compared the performance of the TOA vs. the index-based COA approaches. Both approaches are implemented on the MOD system Hermes, presented in [14]. We used two different granularities to partition the spatial and the temporal hierarchies; a spatial grid of equally sized squares of 10×10 Km^2 (100×100 Km^2, respectively) and a time interval of one (six, respectively) hours. The results of the four cases are illustrated in Figure 13, where it is clear that the choice of a particular method is a trade-off between the selected granularity level and the number of trajectories.

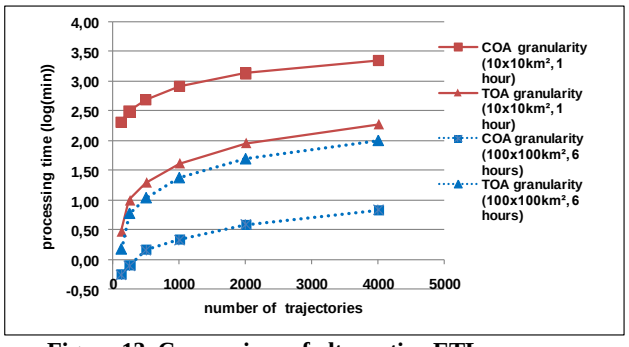

Figure 13. Comparison of alternative ETL processes

We complete the experimental study with some results on the trajectory aggregation issue (Figure 14). We would like to assess the accuracy of the approximations of the measure COUNT_TRAJECTORIES computed in roll-up operations by using the distributive and the algebraic functions presented in Subsection 3.3. To this aim we consider the normalized absolute error proposed by Vitter et al. [18]: For all the OLAP queries q in a set Q we define this error as follows:

$$Error = \frac{\sum_{q \in Q} |\overline{Mq} - Mq|}{\sum_{q \in Q} Mq}$$

where $\overline{Mq}$ is the approximate measure computed for query q, while Mq is its exact value.

We assume, as base granularity g, a spatial grid of equally sized squares of 10×10 Km^2 and a time interval of one hour. Then our queries compute the measure TRAJECTORIES for larger granularities $g' = n \times g$, with $n > 1$.

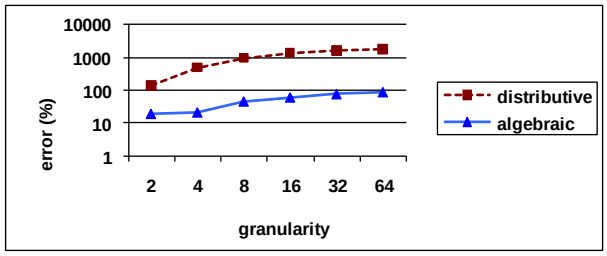

Figure 14. Distributive vs. algebraic aggregate functions

The distributive aggregate function has an error which always exceeds 100% and quickly grows as the roll-up granularity increases. Instead, as expected, the computations based on the algebraic function are always more precise than those based on the distributive one and they are accurate for small granularities. Still, the error grows up for large granularities but it never exceeds 100%. Although the corresponding experiments are not reported here, it is worth noting that starting from smaller base granularities g and using the algebraic function we get a better accuracy, with errors under 10% for small multiples of g.

5. CONCLUSIONS

In this paper, we propose solutions for the efficient and effective development of trajectory warehouses. To the best of our knowledge, this is the first work that supports all the required steps for building a TDW, from trajectory reconstruction and MOD loading, to data cube feeding and aggregating over summary information. More specifically, we proposed techniques for the solution of the *trajectory reconstruction* problem, for supporting *ETL* of trajectory data, and for addressing the problem of measure aggregation, giving particular attention to the *distinct count problem*. Our approach has been experimentally tested in a large real dataset and has been shown to be efficient.

As part of our future work, we plan to examine new measures for the trajectory warehouse, specifically suited for trajectories. An example of such a measure is the so-called *typical trajectory* (e.g. [4], [9]) that describes the trend of movement within a cell. Also, we plan to explore the analytical capabilities of the proposed framework by applying DM techniques over the aggregated data stored in the TDW.

6. ACKNOWLEDGMENTS

We thank S. Orlando, A. Roncato and C. Silvestri for the insightful discussions on the distinct count problem. Research partially supported by the EU FP6-14915 IST/FET Project GeoPKDD (Geographic Privacy-aware Knowledge Discovery and Delivery). Gerasimos Marketos was also supported by a PENED'2003 grant funded by the General Secretariat for Research and Technology of the Greek Ministry of Development.

7. REFERENCES

[1] Agarwal, S., Agrawal, R., Deshpande, P., Gupta, A., Naughton, J., Ramakrishnan, R., and Sarawagi. S. On the computation of multidimensional aggregates. Proc. VLDB, 1996.

[2] Choi, W., Kwon, D., and Lee, S. Spatio-temporal data warehouses using an adaptive cell-based approach. DKE, 59(1):189-207, 2006.

[3] eCourier.co.uk dataset, http://api.ecourier.co.uk/. (URL valid on May 14, 2008).

[4] Giannotti, F., Nanni, M., Pinelli, F., and Pedreschi, D. Trajectory pattern mining. Proc. KDD, 2007.

[5] Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., and Pirahesh, H. Data cube: A relational aggregation operator generalizing groub-by, cross-tab and sub-totals. DMKD, 1(1):29-54, 1997.

[6] Güting, R.H., and Schneider, M. Moving Object Databases, Morgan Kaufman Publishers. 2005.

[7] Han, J., Stefanovic, N., and Koperski, K. Selective Materialization: An Efficient Method for Spatial Data Cube Construction. Proc. PAKDD, 1998.

[8] Jensen, C.S., Kligys, A., Pedersen, T.B., Dyreson, C.E., and Timko, I. Multidimensional data modeling for location-based services, The VLDB Journal, 13:1–21, 2004.

[9] Lee, J., Han, J., and Whang, K. Trajectory Clustering: A Partition-and-Group Framework. Proc. SIGMOD, 2007.

[10] Orlando, S., Orsini, R., Raffaetà, A., Roncato, A., and Silvestri, C. Spatio-Temporal Aggregations in Trajectory Data Warehouses. Proc. DaWaK, 2007.

[11] Orlando, S., Orsini, R., Raffaetà, A., Roncato, A., and Silvestri, C. Trajectory Data Warehouses: Design and Implementation Issues. JCSE, 1(2):240-261,2007.

[12] Papadias, D., Kalnis, P., Zhang, J., and Tao, Y. Efficient OLAP Operations in Spatial Data Warehouses. Proc. SSTD, 2001.

[13] Pelekis, N., Raffaetà, A., Damiani, M.-L., Vangenot, C., Marketos, G., Frentzos, E., Ntoutsi, I., and Theodoridis, Y. Towards Trajectory Data Warehouses. Chapter in Mobility, Data Mining and Privacy: Geographic Knowledge Discovery. Springer-Verlag. 2008.

[14] Pelekis, N., Theodoridis, Y., Vosinakis, S. and Panayiotopoulos, T. Hermes - A Framework for Location-Based Data Management. Proc. EDBT, 2006.

[15] Pfoser, D., Jensen, C.S., and Theodoridis, Y. Novel Approaches to the Indexing of Moving Object Trajectories, Proc. VLDB, 2000.

[16] Tao, T., and Papadias, D. Historical Spatio-Temporal Aggregation. ACM TODS, 23(1):61-102, 2005.

[17] Tao, Y., Kollios, G., Considine, J., Li, F., and Papadias, D. Spatio-Temporal Aggregation Using Sketches. Proc. ICDE, 2004.

[18] Vitter, J.S., Wang, M., and Iyer, B. Data Cube Approximation and Histograms via Wavelets. Proc. CIKM, 1998.