Tracking Livestock Movements to Figure out Potentially Infected Farms

Paolino Di Felice
Dipartimento di Ingegneria Industriale, Informazione ed Economia, Università di L'Aquila, L'Aquila, Italy
Email: *paolino.difelice@univaq.it*

Americo Falcone
Dipartimento di Ingegneria Industriale, Informazione ed Economia, Università di L'Aquila, L'Aquila, Italy
Email: *falconeamerico@gmail.com*

***Abstract*—The concern stems from the public health and food safety aspects of animal health, but also from the economic costs that animal disease outbreaks can trigger. Very recently it has been proposed a method ([4]) devoted to discover the farms that may have been infected by an outbreak of a highly infectious disease of livestock subjected to long trips with intermediary stops. Having a reliable list of farms that may be infected is relevant to feed existing farms culling strategies (e.g., [1]). The present paper reports on an effective way to implement the method introduced in [4] based on an emerging software technology.**

***Index Terms*—livestock movements, moving points databases, SQL, animal health, prevention**

I. INTRODUCTION

A primary concern of national and international institutions for animal health (the World Organization for Animal Health is probably the most known institution among the many - http://www.oie.int/en/), is to keep the animal health under control to prevent epidemics of infectious diseases at geographic scale whose negative effects are the need of culling entire livestock farming, with massive economic costs to the farmers, as well as the risk that the disease transmits to the human beings, too (zoonosis).

The issue of controlling the diffusion of highly infectious livestock epidemics is relevant and topical also from the scientific community, as witnessed by the continuous flow of papers that are published. One of the most recent and important contribution among the many is [1], where the authors propose a method of epidemic investigation (called *risk based culling*) that represents an evolution of the so far mostly adopted *ring culling*.

An input data of the risk based culling strategy is the list of infected farms. Unfortunately, in the cases where the animal batches moved in time periods close to the detection of the disease, with intermediary stops in the so-called "parking areas" (a scenario made frequent by the globe scale livestock market), it is utopian to pretend to know *all* the farms which are infected and, consequently, to think of being able to know exactly the geographical areas affected by the outbreaks of the contagion. This state of affairs reduces tremendously the output reliability of any potential software tool based on [1], simply because the correctness of its prediction is subordinated to the degree of adherence to the reality of its input data.

Ref. [4] proposes a method helpful, downstream of the outbreak of cases of livestock disease, to set up a list of farms that could have been infected by sick head of cattle which moved in a period of time "close to" that when the alarm of a sanitary hazard was issued. The method is made up of an *algorithm* (**CHECK**) and a *database* (about farm, livestock, health checks, trips, and parking areas). Our paper is a continuation and, to a large extent, the completion of the research described in [4]. In a nutshell, aim of the present paper is to give substance to the claim that to manage software applications about the control of the diffusion of the animal diseases more easily, effectively and efficiently than the case where a DataBase Management System of the current generation is adopted, it is necessary making recourse to the body of knowledge about *moving objects databases*, [7].

The paper is structured as follows. Sec. II reports a minimum nucleus of information, taken from [4], necessary to comprehend our contribution. In detail, we sketch out the application context our study refers to and mention the causes that could trigger livestock contagion within the parking areas. Then, it is recalled the algorithm **CHECK** suitable to detect potentially infected batch of animals and, hence, the (potentially) infected farms. The structure of a relational database suitable to model the reference application context ends the section. Sec. III and Sec. IV concern, in sequence, what in [4] was left as "Further work", that is: a) the loading of the designed database with an example dataset; and b) the implementation of the algorithm **CHECK** in terms of SQL queries. As DBMS we use SECONDO [8]. Sec. V ends the paper.

II. THE APPLICATION CONTEXT. ITS MODELLING

doi:10.4304/jait.4.2.101-110

A. Terminology

Hereafter, we use the following terminology mainly inspired by regulations in force within the EU:

- *meeting point*: either the livestock aggregation place during a cattle fair or the terminal where it takes place the loading (unloading) of live animals from a transportation means (truck, ship, and so on).
- *Control post*: a place devoted to the animal nutrition and rest during long trips.
- *Farm*: the place where animals grow up.
- *Parking area*: either a *meeting point*, a *control post*, or a *farm*.
- *Batch*: a certain number of animals of the same species that move together and that, together, rest in the same parking area.
- *Sick batch*: a batch of animals where at least one head of cattle has been found sick after a veterinary visit. We call *potentially infected batch* one containing some head of cattle that might have got the disease from a sick batch.
- *Infected farm*: the farm to which a sick batch belongs. We call *potentially infected farm* one where there is at least a potentially infected batch.

Ref. [6] reports of 157 approved control posts within 12 countries at the beginning of 2010. About meeting points and farms it is not easy to get exact numbers, however they can be estimated of the order of thousands.

Parking areas may be conceptualized as structures either fixed (as in the case of farms) or semi-mobile (as in the case of meeting points) composed of a certain number of pens (Fig. 1).

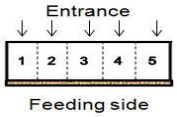

Figure 1. The organization of a parking area made up of 5 pens.

B. Causes Triggering the Contagion. Types of Contagion

Ref. [6] reports that about 365 million farm animals per year are transported within Europe and a large part of them pass through parking areas where they are unloaded and loaded many times before reaching the final destination. The long stops, inside those areas, of the livestock contribute considerably to the diffusion of the epidemics in a short time interval and over large geographical areas.

The causes that could spark off the disease are to be re-conducted to either the "co-presence" in the same parking area of healthy livestock batches and sick ones, or to their "temporal contiguity" (that is when a healthy batch enters a parking area that previously had hosted a sick batch and where, therefore, could have been left few pathogen agents in the environment.). Such two "dimensions" set up the necessary condition because the transmission of the disease among the livestock can take place. Accordingly, two types of contagion have to be taken into account in such an application context: one due to the co-presence (hereafter called *contagion by co-presence*) and the other due to the temporal contiguity (hereafter called *contagion by temporal contiguity*). [4] discusses in some depth both types of contagion.

C. An Investigation Algorithm

Ref. [4] proposes an algorithm (**CHECK**) that, downstream of the identification of an infected batch, traces back to *all* the potentially infected batches of animals, then it recognizes *all* the farms that are to be considered either infected or potentially so. This latter step is fundamental because its output allows to feed the existing methods for the analysis of the diffusion of the disease among farms such as, for instance, the already mentioned risk based culling method. The **CHECK** algorithm follows.

Algorithm **CHECK**

Input: data about the farms, the animal batches, the veterinary visits, the animal trips over the territory, and the involved parking areas.

Output: the (potentially) infected farms

Method:

Let <**SickBatch**, **VisitTimestamp**, **LastVisitFarm**> be, respectively, the identifier of the sick batch, the time stamp when the disease was diagnosed, and the farm where the visit took place.

1. Starting from **LastVisitFarm** and travelling back in time:
   - reconstruct the movements of the sick batch until the farm where it was previously visited (**PreviousVisitFarm**) resulting in healthy is reached. Both the **PreviousVisitFarm** and the **LastVisitFarm** are assumed to be infected.
     Let {**PreviousVisitFarm**, $\mathbf{PA_1}$, $\mathbf{PA_2}$, …, $\mathbf{PA_k}$, **LastVisitFarm**} be the result of this investigation step, where $\mathbf{PA_i}$ (with **i=1,2,3,…,k**) denotes the generic parking area that had put the sick batch up.
   - For each $\mathbf{PA_i}$, compute the duration of the stop of the **SickBatch** in it and the departure time from it
2. for each $\mathbf{PA_i}$, the issue is to identify the animal batches that might have been infected by the **SickBatch**;
3. for each those batches, identify (*when possible*) the farms they belong to, these latter to be classified as potentially infected too.

Notice that when the **CHECK** algorithm is started, not necessarily it happens that all the batches returned by Step "2." have reached the destination farm. Some of them, in fact, could be still (away) on the trip towards the final destination. This is the meaning of the words "*when possible*". For each animal batch potentially infected by a **SickBatch** that falls in such

a situation, Step "3." returns the last parking area occupied by the livestock.

D. A SECONDO database

Di Felice and Falcone ([4]) complete their proposal by designing a SECONDO relational database about farm, livestock, health checks, trips, and different types of parking areas as an essential step towards an effective and efficient implementation of the **CHECK** algorithm. Their database is composed of the following five tables:

```
animalBatch (BatchId: string,
    Species: string, HeadNumber: int);
parkingArea (PAID: string, Name:
    string, City: string, Type:
    string, FarmerId: string,
    Position: point, Layout: region);
farmer (FarmerId: string,
    Name: string);
visit (BatchId: string, VisitDate:
    instant, Result: string,
    Diagnosis: string, PAID: string);
trip (BatchId: string, TripData:
    mpoint, From: string, To: string);
```

The APPENDIX collects the SECONDO SQL-like definition of those tables.

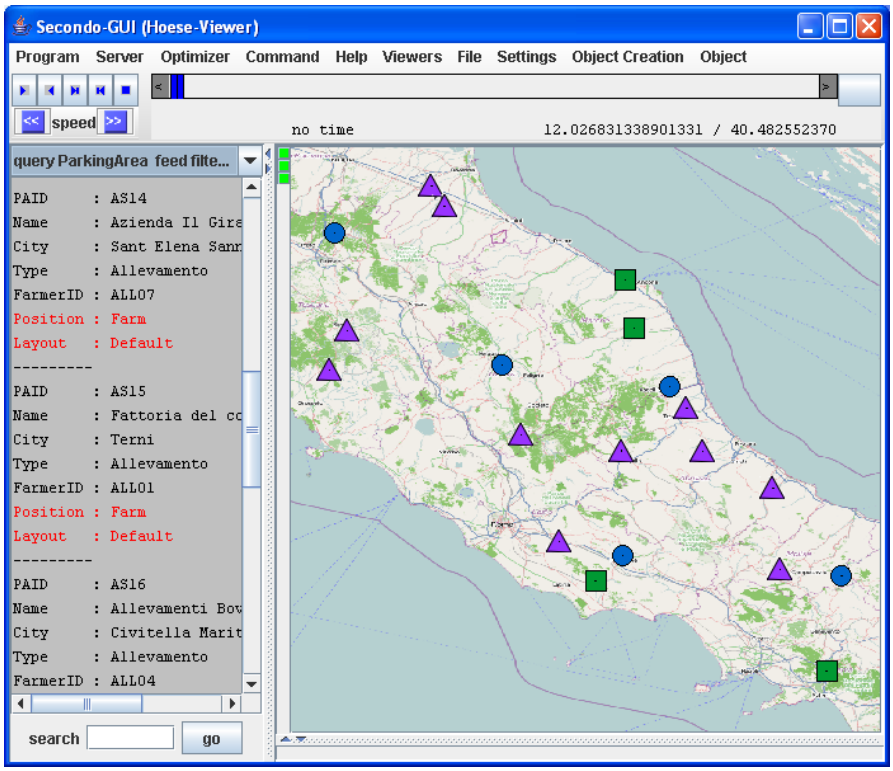

Figure 2. The map of the parking areas of the example dataset

This database models the movement of an animal batch (from a parking area to another one) as an atomic value of the attribute **`TripData`** of type **`mpoint`**, [7]. The organization of the database in terms of moving points (briefly *m-points*) acknowledges the recent recommendations of the EU ([2] Annex I, Chp.VI, Point 4 – *Navigation System*) which hope a prompt activation of a fully electronic procedure about the traceability of the movements of live animals (see, for instance, the "Identification and Tracing" section of the Animal Health Strategy of the European Union - 2007-2013, [5]).

III. AN EXAMPLE DATASET

We have loaded the database of Sec. IID with an example dataset small but still sufficiently comprehensive to cover the cases of contagion between lots of cattle recalled in Sec IIB. The dataset consists of 20 animal batches and 20 parking areas (located in the centre of Italy – Fig. 2) broken down as follows: 11 farms (the violet triangles), 5 control posts (the blue circles) and 4 meeting points (the green squares).

The APPENDIX collects a summary of the SQL-like scripts about the loading of the tuples into the SECONDO database.

The map shown in Fig. 2 is the combined output of the processing of the following three queries:

```
SELECT *
FROM parkingArea
WHERE type = "Meeting point"

SELECT *
FROM parkingArea
WHERE type = "Control post"

SELECT *
FROM parkingArea
WHERE type = "Farm"
```

by defining for each query a different visualization display of the spatial objects (i.e., of each geometric attribute of the relation **`parkingArea`**).

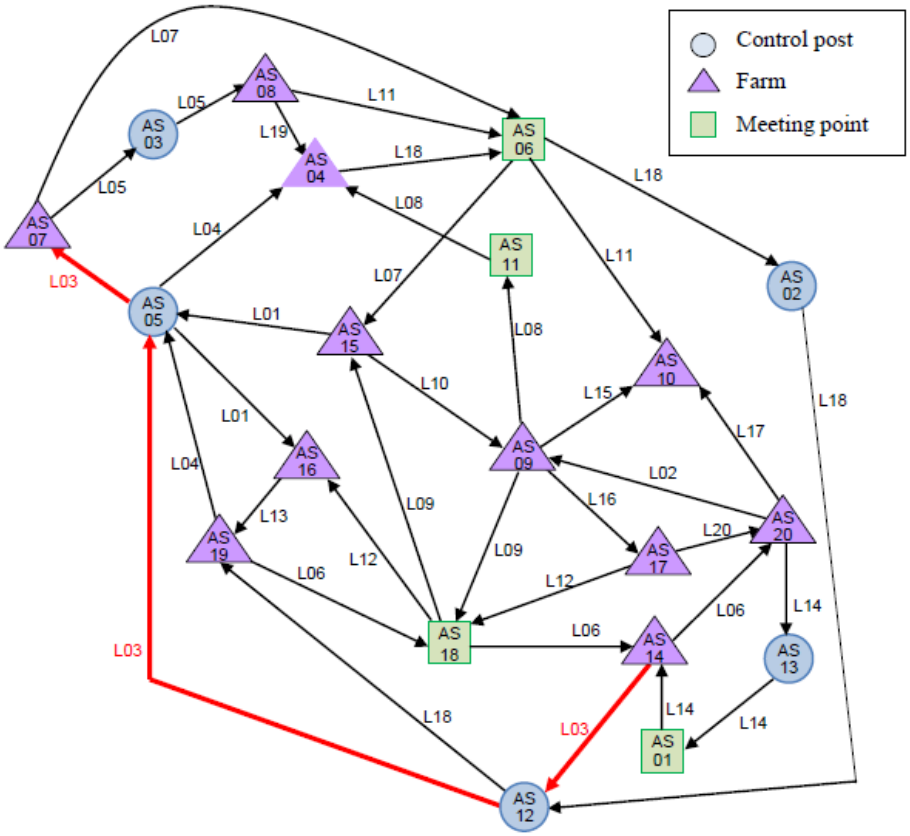

Figure 3. The graph of the movements of the animal batches being part of the example dataset

The database contains also the data of 37 trajectories corresponding to as many trips of the batches between pairs of parking areas. The movements of the animals have been generated by a Java program which receives as input the parking area of departure and arrival, the date and start time of the trip and returns a text file that describes the journey. These movements are conceptualized by the graph of Fig. 3, where each node is labeled with the code of a parking area (namely, a string ranging from AS01 to AS20), while the arcs are labeled with the code of the batch that moved between the extreme nodes. As we can see, different batches have gone through the same intermediate parking areas before reaching the final destination, a circumstance very common in the reality.

The (red) arcs labeled L03 in Fig. 3 refer to the animal batch L03 that, as the result of a transaction, moved from farm AS14 to farm AS07. Before departure, the livestock was subjected to a veterinary check at the farm of origin (AS14) with negative outcome. Reached the destination (AS07), on 11/07/2011 the batch was visited again by resulting sick (see Sec. IIA for the definition of "sick batch") of a disease highly infectious. By construction, L03 is the only one sick batch in our small example dataset.

Going through the steps of the algorithm **CHECK** for the example dataset, and taking into account the arrival and departure time of the livestock from the parking areas, we get the situation depicted in Fig. 4 and summarized in Table 1.

IV. IMPLEMENTATION OF THE ALGORITHM CHECK

This section reports on the implementation of the algorithm CHECK. The solution is valid independently of the number of sick batches. **CHECK** has been realized in terms of eight "basic" queries. Table 2 shows the correspondence that exists between them and the steps of the algorithm. It is trivial to reduce the number of queries simply by "merging" the basic queries of the same level at the expense, however, of a greater difficulty of understanding the resulting queries.

In the following, the term query is overloaded in the sense that it denotes both what we want to compute and the SQL formulation to reach the goal.

The syntax of the eight queries most adhere to standard SQL. The few variations will be explained as we met them.

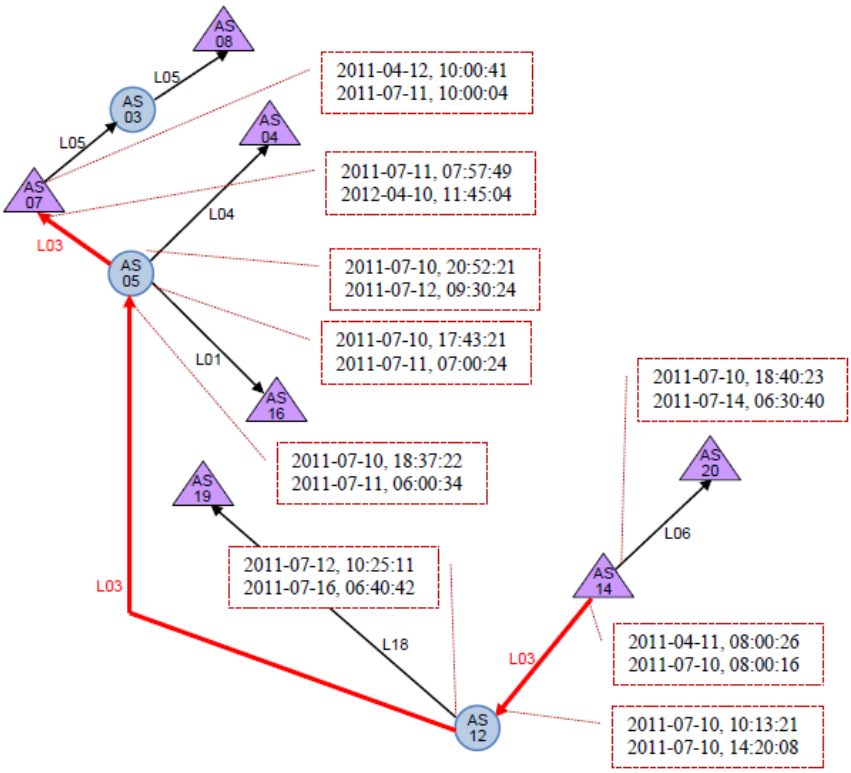

Figure 4. The portion of the graph of Fig. 3 that plays an active role in discovering (potentially) infected farms. The rectangles contain the time stamp of arrival and departure of the livestock from the parking area. "yyyy-mm-dd, hh:mm:ss" is the date-time format used. For example, L03 entered AS12 on 2011-07-10 at 10:13:21 and left it on 2011-07-10 at 14:20:08.

Table 1. **CHECK** outcome with respect to the example dataset. To verify the table content correctness, please refer to the time stamp values shown in Fig. 4.

Sick batches	L03	
Potentially infected batches		
	BatchID	Place of contact
	L01	AS05 (farm)
	L04	AS05 (control post)
	L05	AS07 (farm)
	L06	AS14 (farm)
	L18	AS12 (control post)
Infected farms	AS14, AS07 (because of the presence of L03)	
Potentially infected farms		
	Farm	Infecting batch
	AS04	L04
	AS08	L05
	AS16	L01
	AS19	L18
	AS20	L06

Table 2. Correspondence between **CHECK** and the basic queries

Steps of **CHECK**	Queries that implement the step
1	1, 2, 3, 4, 5
2	6, 7
3	8

A. *Implementation of Step 1*

Preliminarily we determine the periods spent by the animal batches in the parking areas (**Query 1**, **2** and **3**). Those data are extracted from the trajectories and stored into a working table (**`break`**). Three cases are possible (Fig. 5), each implemented as an independent query, according to the "role" played by the parking area inside the whole history of the movements of each animal batch present in the database. Let us denote with *bid** and *paid**, respectively, the identifier of a generic animal batch and that of a generic parking area. The three roles played by *paid** are the following:

- *a crossing parking area for bid** (Case "a" of Fig. 5). In the database exists *at least* a trajectory done by *bid** that reaches *paid**, stops in it, and then leaves from it.
- *The last known destination of bid** (Case "b"). In the database exists a trajectory done by *bid** that reaches *paid**, but none comes out.

– *The origin parking area for bid** (Case "c"). In the database exists a *trajectory* done by *bid** that comes out from *paid**, but none enters it. Think, for example, to *paid** as the farm of birth of *bid**.

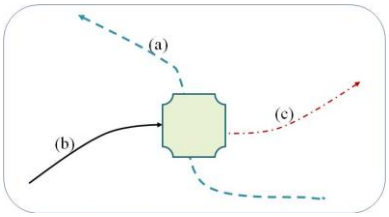

Figure 5. The roles played by a parking area in the history of movements of an animal batch

Query 1: Rest periods of the batches inside crossing parking areas

```
let break =
SELECT [entry:BatchId AS BatchNumber,
    entry:To AS ParkingAreas,
    inst (final (entry:TripData))
      AS StartStaging,
    inst (initial (exit:TripData))
      AS EndStaging]
FROM  [trip AS entry, trip AS exit]
WHERE [entry:BatchId = exit:BatchId,
       entry:To = exit:From]
```

The (SECONDO) `let` command builds the table `break`, which stores the query result. `entry:BatchId` replaces the `entry.BatchId` standard notation.

The beginning instant of the rest in a parking area coincides with the last instant of the incoming trajectory in such an area, while the end of the rest is the initial instant of the trajectory of output from the same area. "`inst(final(entry: tripData))`" and "`inst(initial(exit: tripData))`" return, respectively, those time stamps (`entry` and `exit` are two aliases of the `trip` table).

Query 2: Rest periods of the batches inside the parking area of their last known destination

```
INSERT into break
SELECT [entry:BatchId AS BatchNumber,
    entry:To AS ParkingAreas,
    inst (final (entry:TripData))
          AS StartStaging,
    now AS EndStaging]
FROM  [trip AS entry]
WHERE [entry:to NOT IN
         (SELECT t:From
          FROM   [trip AS t]
          WHERE  entry:BatchId =
                   t:BatchId )]
```

The stay period in the "last known destination" returned by **Query 2** ranges from the time of the last timestamp of the input trajectory in the parking area and the time of execution of query itself (in SECONDO, the date and current time are returned by the operator `now`).

Query 3: Stay periods of the batches inside their origin parking area

```
INSERT into break
SELECT [exit:BatchId AS BatchNumber,
    exit:From AS ParkingAreas,
        inst (initial(exit:TripData )) -
          [const, duration, value,
          [90,0]] AS StartStaging,
        inst (initial(exit:TripData ))
          AS EndStaging]
FROM  [trip AS exit]
WHERE [exit:From NOT IN
         (SELECT t:To
          FROM   [trip AS t]
          WHERE  exit:BatchId =
                   t:BatchId )]
```

The parameter "`[const, duration, value, [90,0]]`" sets the temporal extension of the stay period (90 days) in an "origin parking area" returned by **Query 3**. Such a value can be modified according to the needs.

Query 4: Migration of the `break`'s content into table `stops` and addition, to this latter, of attribute `rangeTime` that stores the stay time interval

```
let stops = break feed
  extend[rangeTime: theRange
      (.StartStaging,
       .EndStaging, true, true)]
  sortby[BatchNumber, StartStaging]
consume;
```

The `feed` operator reads relation `break` from disk and puts its tuples into a stream; while the `extend` operator adds the attribute `rangeTime` to the query result; lastly, the `consume` operation collects a tuple stream into a persistent relation. `.StartStaging` stands for `break.StartStaging`.

Fig. 6 shows a portion of the instance of the relation `stops` computed with respect to the example dataset of Sec. III.

Query 5: Infected farms

```
SELECT [t:BatchId AS SickBatches,
    site:PAID AS InfectedFarms]
FROM   [ParkingArea AS site,
    trip AS t, visit AS v ]
WHERE  [v:result ="Sick",
    t:BatchId = v:BatchId,
    t:TripData passes site:Layout,
    site:Type ="Farm"]
```

Query 5 analyzes the trips of each sick batch, to assess whether they crossed the area that borders some of the farms in the database (predicate: "`t:TripData passes site:Layout`"). Fig. 7 shows the output of **Query 5**. The result coincides with the expectation (see Table 1).

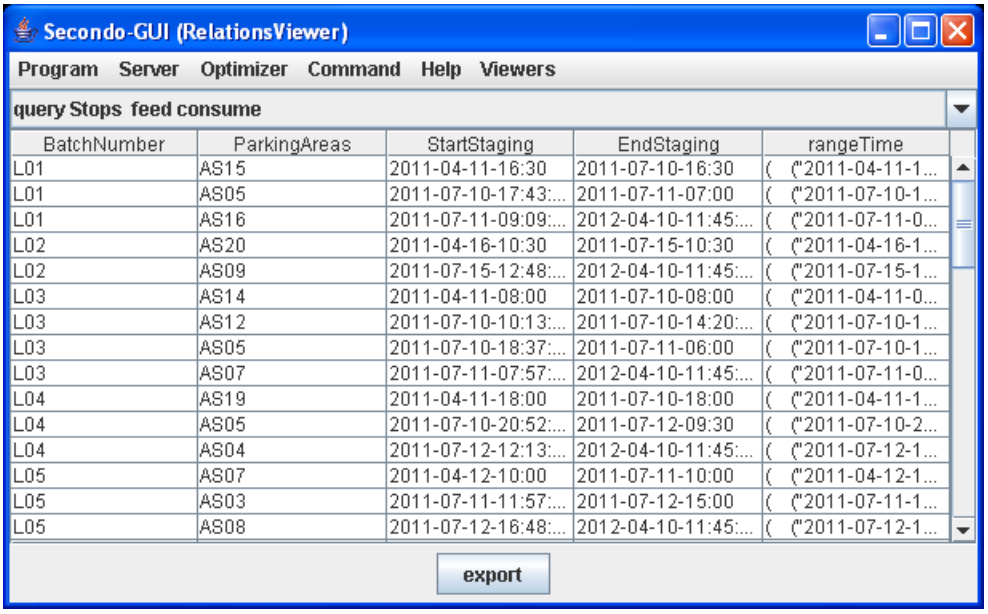

Figure 6. Stay intervals of the animal batches inside the parking areas (i.e., meeting points, control posts, and farms). Giving a glance at the rows of the table, it is possible to have a confirmation, for instance, of the trips of the batch L03 together with the relative time stamps, previously seen in Fig.4.

B. Implementation of Step 2 (Search potentially infected batches)

Query 6 uses the data in the **stops** table to determine the batches potentially infected by the co-presence with batches found sick.

```
# Query 6: Potentially infected batches by co-presence
SELECT [stay1:BatchNumber AS
       SickAnimalBatches,
    stay2:BatchNumber AS
       PotentiallyInfectedBatches,
    stay2:ParkingAreas AS
       SitesOfInfection]
FROM  [stops AS stay1, stops AS stay2,
       visit AS v]
WHERE [tay1:BatchNumber #
       stay2:BatchNumber,
    stay1:ParkingAreas =
       stay2:ParkingAreas,
    stay1:rangeTime intersects
       stay2:rangeTime,
    stay1:BatchNumber = v:BatchId,
       v:result = "Sick"]
ORDERBY [SickAnimalBatches,
    SitesOfInfection,
    PotentiallyInfectedBatches]
```

The tables listed in **Query 6** are **stops** and **visit** The **predicate** "**stay1:BatchNumber = v:BatchId**, **v:result="Sick"**" identify *all* the sick batches in the database. Then, are selected *all* the animal batches who have made stops in the same parking area (predicate: "**stay1:ParkingAreas = stay2:ParkingAreas**") by ignoring the tuples that refer to the same batch (predicate: "**stay1:BatchNumber # stay2:BatchNumber**"). Lastly, the function **intersects** verifies the temporal overlapping of their periods of stay. The **SELECT** clause lists the columns to be displayed, namely: the ID of the sick batch, the infected batches, the parking area where the infection could be occurred, the range of co-presence.

The formulation of **Query 6** is valid regardless of the number of sick batches in the database. This thanks to the condition "**v:result = "Sick"**" which takes into account all the sick batches. In the example dataset there is only one sick batch (L03).

With a similar procedure, it is possible to determine the batches infected by temporal contiguity and the places where such a contamination may have occurred (**Query 7**).

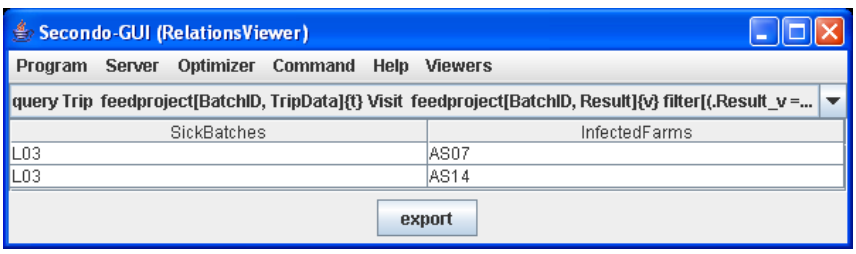

Figure 7. The farms infected by the batch L03

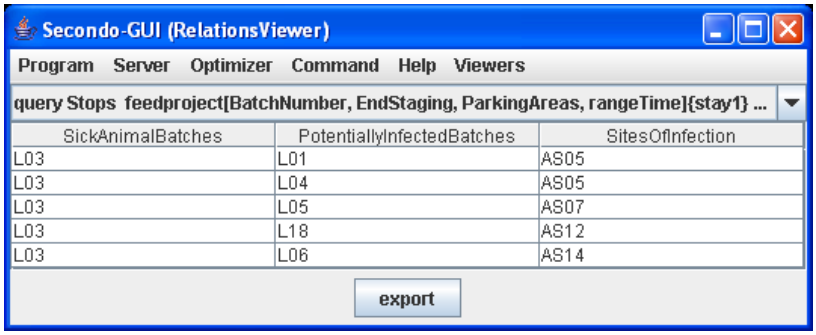

Figure 8. Batches potentially infected by batch L03 either by co-presence or temporal contiguity

Query 7: Potentially infected batches by temporal contiguity

```
SELECT [stay1:BatchNumber AS
        SickAnimalBatches,
    stay2:BatchNumber AS
        PotentiallyInfectedBatches,
    stay2:ParkingAreas AS
        SitesOfInfection]
FROM  [stops AS stay1, stops AS stay2,
        visit AS v]
WHERE [stay1:BatchNumber #
    stay2:BatchNumber,
stay1:ParkingAreas =
    stay2:ParkingAreas,
stay1:BatchNumber = v:BatchId,
    v:result ="Sick",
    stay2:StartStaging >
        stay1:EndStaging,
    stay2:StartStaging <
        stay1:EndStaging +
        [const, duration, value, [2,0]]]
```

Because it exists temporal contiguity, the beginning of the rest of a batch must start after the end of the stay of the sick one (**predicate**: "`stay2:StartStaging > stay1:End Staging`"). In **Query 7**, the parameter "`[const, duration, value, [2,0]]`" sets the "temporal distance" between these two events in two days. The value of such a parameter *have to be* changed according to the characteristics of the epidemic at hand.

Query 6 and **Query 7** can be merged in a single query by using the `or` operator in the `WHERE` clause. The combined effect of these two queries is shown in Fig. 8.

C. Implementation of Step 3 (Search potentially infected farms)

Query 8 returns the potentially infected farms, that is the farms which host at least one of the batches potentially infected (either by co-presence or temporal contiguity) by the sick batches inside some of the parking areas. The screen of Fig. 9 shows the result.

#**Query 8**: Potentially infected farms

```
SELECT [stay1:BatchNumber AS
        SickBatches,
    stay2:BatchNumber AS
        PotentiallyInfectedBatches,
    lastTrip:to AS
        PotentiallyInfectedFarms]
FROM  [stops AS stay1, stops AS stay2,
    trip AS lastTrip, visit AS v ]
WHERE [stay1:BatchNumber #
        stay2:BatchNumber,
    stay1:ParkingAreas =
        stay2:ParkingAreas,
    stay1:BatchNumber = v:BatchId,
    v:result ="Sick",
    (stay1:rangeTime intersects
        stay2:rangeTime )
    or
    (stay2:StartStaging <
        stay1:EndStaging +
        [const, duration, value, [2,0]]
    and
    stay2:StartStaging >
        stay1:EndStaging ),
    lastTrip:BatchId =
        stay2:BatchNumber,
    lastTrip:to
    NOT IN
    (SELECT [journey:From]
     FROM  [trip AS journey]
     WHERE [stay2:BatchNumber =
        journey:BatchId ])]
ORDERBY[SickBatches,
     PotentiallyInfectedFarms]
```

Fig. 10 summarizes the outcome of the analysis (output of **Query 5** and **Query 8**) on a geographic map, that is the infected farms (red crosses) and those potentially infected (triangles with an embedded exclamation mark).

V. CONCLUSIONS

The paper reports about the implementation of a method (the algorithm **CHECK**) that takes advantage of the data collected in a "quasi real-time" database about the trips of the livestock from a parking area to another one and the sanitary controls of the livestock itself, in order to derive which farms are infected and which one could be so. The availability of this latter information allows to feed the existing methods for the analysis of the diffusion of the disease among farms such as, for instance, the risk based culling. The proposed solution cuts the number of head of cattle on which has to be launched the campaign of visits, that

otherwise should be extended to *all* the livestock which has undergone movements in the period of time elapsed from the visit of the head turned out to be sick and the previous visit, in which the same animal was healthy.

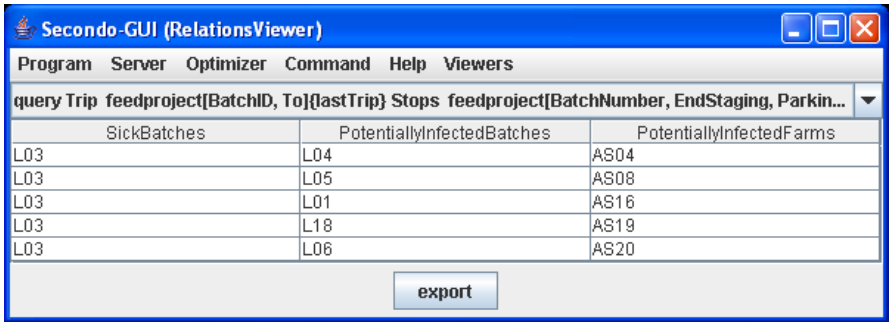

Figure 9. The potentially infected farms

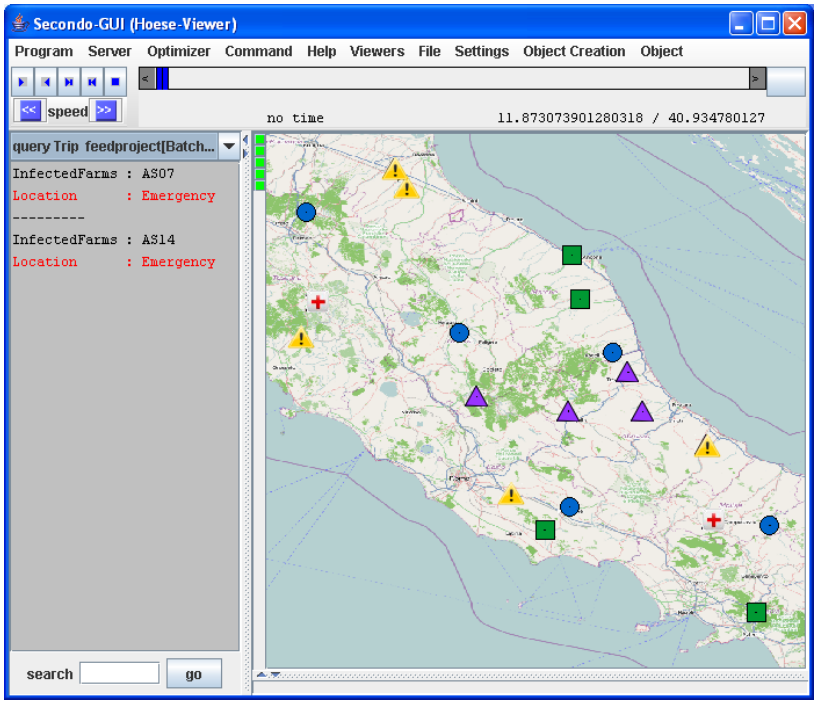

Figure 10. The outcome of the **CHECK** algorithm against the example dataset

The happy notes learned through the experience

The implementation in SECONDO of the algorithm **CHECK** has been accomplished in terms of eight SQL queries of low difficulty. The realization effort has to be considered, therefore, within the reach of anyone who wants to repeat of his own a solution such as that reported in this paper. Incomparably bigger is the entity of the effort if one decides to adopt as enabling technology one of the RDBMS today available on the marketplace (e.g.: IBM-DB2/SE, Oracle Spatial, or PostgreSQL/PostGIS) and this for the lack in those software of a native data type suitable to model moving points and, consequently, of operators that operate on those complex objects ([3] discusses this issue in detail).

Without such a native support, the implementation of the algorithm **CHECK** binds us to develop, in advance, ad hoc operators (such as, for example, passes used in **Query 5**) with a global effort definitely higher, besides the risk of producing a software of lower reliability.

The painful notes learned through the experience

So far, SECONDO cannot be considered a stable technology to put into practice in real contexts. This system, to the authors' own admission, it is now recommended especially in the scientific context mainly for testing new methods and algorithms.

To reach a satisfactory command in the use of SECONDO, it requires a period of start-up absent if one remains with the relational DBMSs today largely part of most corporate assets.

REFERENCES

[1] D. E. te Beest, T. J. Hagenaars, A. J. Stegeman, M. PG Koopmans, and M. van Boven, "Risk based Culling for highly Infectious Diseases of Livestock," *Veterinary Research* 2011, 42:81.

[2] Council Regulation (EC) No 1255/97 of 25 June 1997 "On Community Criteria for Staging Points and Amending the Route Plan Referred to in the Annex to Directives 91/628/EEC," *Official Journal of the European Union* series L, n. 174/1, 2.07.1997. (http://europa.eu/documentation/legislation/index_en.htm - item: Search by Official Journal reference, June 2012)

[3] P. Di Felice. A Short Term Solution to Implement Applications about Moving Points on top of Existing DBMSs. *Int. Journal of Computer Applications* (0975 – 8887) Volume 50 – No.10, July 2012. DOI: 10.5120/7804-0934.

[4] P. Di Felice, A. Falcone. "An Algorithm and a Database: two Conceptual Tools to Control the Diffusion of Animal Diseases," *Journal of Advances in Information Technology*. (*To appear*)

[5] European Commission, The new Animal Health Strategy for the European Union (2007-2013): "Prevention is better than Cure," *Communication from the Commission to the Council, the European Parliament, the European Economic and Social Committee and the Committee of the Regions* - COM 539 (2007). European Communities, 2007. ISBN 978-92-79-06722-8. (http://ec.europa.eu/food/animal/diseases/strategy/index_en.htm, June 2012)

[6] G. Gebresenbet, W. Baltussen, P. Sterrenburg, K. De Roest, K. E. Nielsen, "Evaluation of the Feasibility of a Certification Scheme for high Quality Control Posts," Sanco/d5/2005/SI2.548887, 2010. *European Commission Funded Project Directorate-General for Health and Consumers*.

[7] R. H. Güting, M. Schneider, *Moving Objects Databases*, Morgan Kaufmann Publishers, 2005.

[8] R. H. Güting, T. Behr, and C. Düntgen, "SECONDO: A Platform for Moving Objects Database Research and for Publishing and Integrating Research Implementations," *IEEE Data Engineering Bulletin*, 2010, 33:2, 56-63.

APPENDIX

This section collects a summary of the SECONDO scripts devoted to create the database, its tables and load them with the example dataset.

```
# DB creation and opening
create database MODAT;
# Moving Objects Database for Animal Traceability
open database MODAT;

# Tables creation
  sql CREATE TABLE animalBatch COLUMNS (BatchID: string,
      Species: string, HeadNumber: int )
  sql CREATE TABLE parkingArea COLUMNS (PAID: string, Name:
      string, City: string, Type: string, FarmerID: string, Position:
      point, Layout: region )
  sql CREATE TABLE farmer COLUMNS (FarmerID: string, Name:
      string )
  sql CREATE TABLE visit COLUMNS (BatchID: string, VisitDate:
      instant, Result: string, Diagnosis: string, PAID: string)

# Tables loading (partial)

# Animal batch (1 of 20)
sql insert into animalBatch values ("L01", "Bovina chianina", 30)

# Parking area
sql insert into parkingArea values ("AS01", "Cerullo s. r. l.",
    "Montoro Superiore", "Meeting point", "ALL02",
    [const, point, value, (14.7949, 40.8512)],
    [const, region, value, (((
    (14.79435 , 40.85105), (14.79505, 40.85090),
    (14.79590 , 40.85100), (14.79590, 40.85150),
    (14.79545 , 40.85195), (14.79480, 40.85165),
    (14.79435 , 40.85165)))) ]

# Farmer (1 of 8)
sql insert into farmer values ("ALL01", "Mario Bramieri")

# Visit (1 of 11)
sql insert into visit values ("L01", theInstant (2011,07,2,11,00),
"Regolare", " Healthy", "AS15")

# Trip creation and loading
```

The creation and the loading of the table **`trip`** require several steps. The reader interested to know the details may refer to the *Appendix* in [3].

Call for Papers and Special Issues

Aims and Scope

JAIT is intended to reflect new directions of research and report latest advances. It is a platform for rapid dissemination of high quality research / application / work-in-progress articles on IT solutions for managing challenges and problems within the highlighted scope. JAIT encourages a multidisciplinary approach towards solving problems by harnessing the power of IT in the following areas:

- **Healthcare and Biomedicine** - advances in healthcare and biomedicine e.g. for fighting impending dangerous diseases - using IT to model transmission patterns and effective management of patients' records; expert systems to help diagnosis, etc.
- **Environmental Management** - climate change management, environmental impacts of events such as rapid urbanization and mass migration, air and water pollution (e.g. flow patterns of water or airborne pollutants), deforestation (e.g. processing and management of satellite imagery), depletion of natural resources, exploration of resources (e.g. using geographic information system analysis).
- **Popularization of Ubiquitous Computing** - foraging for computing / communication resources on the move (e.g. vehicular technology), smart / 'aware' environments, security and privacy in these contexts; human-centric computing; possible legal and social implications.
- **Commercial, Industrial and Governmental Applications** - how to use knowledge discovery to help improve productivity, resource management, day-to-day operations, decision support, deployment of human expertise, etc. Best practices in e-commerce, e-commerce, e-government, IT in construction/large project management, IT in agriculture (to improve crop yields and supply chain management), IT in business administration and enterprise computing, etc. with potential for cross-fertilization.
- **Social and Demographic Changes** - provide IT solutions that can help policy makers plan and manage issues such as rapid urbanization, mass internal migration (from rural to urban environments), graying populations, etc.
- **IT in Education and Entertainment** - complete end-to-end IT solutions for students of different abilities to learn better; best practices in e-learning; personalized tutoring systems. IT solutions for storage, indexing, retrieval and distribution of multimedia data for the film and music industry; virtual / augmented reality for entertainment purposes; restoration and management of old film/music archives.
- **Law and Order** - using IT to coordinate different law enforcement agencies' efforts so as to give them an edge over criminals and terrorists; effective and secure sharing of intelligence across national and international agencies; using IT to combat corrupt practices and commercial crimes such as frauds, rogue/unauthorized trading activities and accounting irregularities; traffic flow management and crowd control.

The main focus of the journal is on technical aspects (e.g. data mining, parallel computing, artificial intelligence, image processing (e.g. satellite imagery), video sequence analysis (e.g. surveillance video), predictive models, etc.), although a small element of social implications/issues could be allowed to put the technical aspects into perspective. In particular, we encourage a multidisciplinary / convergent approach based on the following broadly based branches of computer science for the application areas highlighted above:

Special Issue Guidelines

Special issues feature specifically aimed and targeted topics of interest contributed by authors responding to a particular Call for Papers or by invitation, edited by guest editor(s). We encourage you to submit proposals for creating special issues in areas that are of interest to the Journal. Preference will be given to proposals that cover some unique aspect of the technology and ones that include subjects that are timely and useful to the readers of the Journal. A Special Issue is typically made of 10 to 15 papers, with each paper 8 to 12 pages of length.

The following information should be included as part of the proposal:

- Proposed title for the Special Issue
- Description of the topic area to be focused upon and justification
- Review process for the selection and rejection of papers.
- Name, contact, position, affiliation, and biography of the Guest Editor(s)
- List of potential reviewers
- Potential authors to the issue
- Tentative time-table for the call for papers and reviews

If a proposal is accepted, the guest editor will be responsible for:

- Preparing the "Call for Papers" to be included on the Journal's Web site.
- Distribution of the Call for Papers broadly to various mailing lists and sites.
- Getting submissions, arranging review process, making decisions, and carrying out all correspondence with the authors. Authors should be informed the Instructions for Authors.
- Providing us the completed and approved final versions of the papers formatted in the Journal's style, together with all authors' contact information.
- Writing a one- or two-page introductory editorial to be published in the Special Issue.

Special Issue for a Conference/Workshop

A special issue for a Conference/Workshop is usually released in association with the committee members of the Conference/Workshop like general chairs and/or program chairs who are appointed as the Guest Editors of the Special Issue. Special Issue for a Conference/Workshop is typically made of 10 to 15 papers, with each paper 8 to 12 pages of length.

Guest Editors are involved in the following steps in guest-editing a Special Issue based on a Conference/Workshop:

- Selecting a Title for the Special Issue, e.g. "Special Issue: Selected Best Papers of XYZ Conference".
- Sending us a formal "Letter of Intent" for the Special Issue.
- Creating a "Call for Papers" for the Special Issue, posting it on the conference web site, and publicizing it to the conference attendees. Information about the Journal and Academy Publisher can be included in the Call for Papers.
- Establishing criteria for paper selection/rejections. The papers can be nominated based on multiple criteria, e.g. rank in review process plus the evaluation from the Session Chairs and the feedback from the Conference attendees.
- Selecting and inviting submissions, arranging review process, making decisions, and carrying out all correspondence with the authors. Authors should be informed the Author Instructions. Usually, the Proceedings manuscripts should be expanded and enhanced.
- Providing us the completed and approved final versions of the papers formatted in the Journal's style, together with all authors' contact information.
- Writing a one- or two-page introductory editorial to be published in the Special Issue.

More information is available on the web site at http://www.academypublisher.com/jait/.

(Contents Continued from Back Cover)