Cleaning trajectory data of RFID-monitored objects through conditioning under integrity constraints

Bettina Fazzinga, Sergio Flesca, Filippo Furfaro, Francesco Parisi
DIMES, UNICAL, Rende, IT
{bfazzinga,flesca,furfaro,fparisi}@dimes.unical.it

ABSTRACT

A probabilistic framework is introduced for reducing the inherent uncertainty of trajectory data collected for RFID-monitored objects. The framework represents the position of an object at each instant as a random variable over the set of possible locations. The probability density function of this random variable is initialized according to an a-priori probability distribution, and then revised by conditioning it w.r.t. the event that integrity constraints are satisfied. In particular, integrity constraints implied by the structure of the map of locations and the motility characteristics (such as the maximum speed) of the monitored objects are exploited (namely, *direct unreachability*, *latency* and *minimum traveling time* constraints). The efficiency and effectiveness of the proposed approach are assessed experimentally on synthetic data.

1. INTRODUCTION

RFID-based applications. The recent improvements in the RFID technology have led to the pervasive use of RFID devices as a support for object tracking. Basically, RFID technology relies on two types of device, i.e. *tags* (which can emit a radio signal encoding simple identification information) and *readers* (which detect the signals emitted by tags). Thus, stays and movements can be monitored by appropriately placing RFID readers in the locations and then attaching RFID tags to the objects to be tracked.

One of the prominent scenarios which can benefit from the use of RFID technology is the monitoring of people, animals, and objects moving inside buildings [18, 19, 22], such as museums, schools, hospitals, office buildings, factories, farms. The reasons for this kind of monitoring are various, and range from collecting data to support the behavior analysis over the monitored entities, to ensuring security for people and assets. For instance, information on the trajectory followed by monitored people can be used to prevent or look into crimes, and detect dangerous or suspicious situations. As another example, information on the trajectory followed by a visitor inside a museum can be used to provide her during her visit with very detailed context-aware information, that are personalized on the basis of the artworks she saw in the rooms visited previously.

Ambiguity of RFID data. For each monitored object, the collected data are pairs $\langle$*timestamp*, *reader*$\rangle$, or, equivalently, $\langle$*timestamp*, *set of readers*$\rangle$ (called "*readings*" in the following), each encoding the fact that the object was detected by the specified set of RFID readers at the specified time instant. Analysis tasks on the monitored world commonly reason on an interpreted version of these data, obtained by moving from the point of view of readers to that of locations. In particular, object-tracking and trajectory-analysis tools often require the readings to be in the form $\langle$*timestamp*, *location*$\rangle$. The point is that, generally, there is not a one-to-one correspondence between locations and readers, and no way to deterministically decide the location given that a set of readers detected an object: the same location may contain zones where an object can be detected by different readers, the same reader may detect objects at different locations, and false negative readings may happen as well (an object close to a reader is not detected, owing to interferences or malfunction).

For instance, consider Fig. 1(a). If an object o was detected at some instant by both readers r_1 and r_5, two locations are possible: L_1 or L_4. Analogously, if o was detected by r_3 only, we can not conclude that it was surely in L_3, as it could be the case that r_2 failed to detect it despite it was close enough to its antenna. Thus, also the case that o was in L_2 (in particular, in the zone of L_2 where an object can be detected by both r_2 and r_3) is a possibility.

This suggests that the association readers/locations can be naturally modeled in probabilistic terms, for instance by means of a probability distribution $p^a(l|R)$ defined for each location l and set

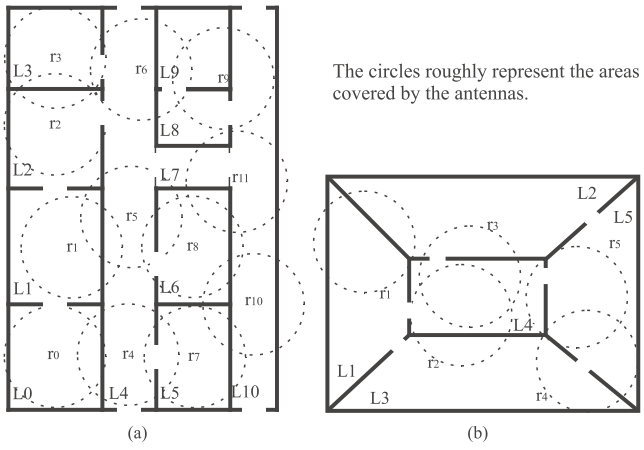

Figure 1: (a) **A floor of a building;** (b) **map used in Example 4 and subsequent ones.**

10.5441/002/edbt.2014.35

R of readers, which represents the probability that an object detected by all and only the readers in R is at location l. In the case depicted in Fig. 1, $p^a(l|R)$ could be such that $p^a(L_1|\{r_1, r_5\}) = p^a(L_4|\{r_1, r_5\}) = 0.5$ and $p^a(L_0|\{r_0\}) = 1$ (for the sake of brevity, we do not report the other combinations location/set of readers). Such a probabilistic model can be obtained in several ways. For instance, by building a "physical" model, which takes into account the placement of the readers over the map and the variability of the reading rate of the readers vs. the distance. An example of this approach is the three-state model proposed in [4]. Otherwise, $p^a(l|R)$ can be defined experimentally, on the basis of samples of detections of tagged objects (this is how we obtained the distributions $p^a(l|R)$ used in our experiments).

Things become more complex when trying to translate a sequence of readings collected for an object o over a time interval into possible trajectories (i.e., sequences of locations). A naive way to solve this problem is to use the knowledge of $p^a(l|R)$ and consider the steps of the trajectory independent from each other, as in the following example.

EXAMPLE 1. *Consider Fig. 1(a) and assume that, at both instants* 0 *and* 1*, object o is detected by both r_1 and r_5, while at instant* 2 *it is detected by r_0. Moreover, assume that $p^a(l|R)$ is as discussed above, that is, objects detected by both r_1 and r_5 are at location L_1 with probability* 0.5 *or at L_4 with probability* 0.5*, and those detected by r_0 are at L_0 with probability* 1*. Reasoning only on the basis of these probabilities and exploiting no further knowledge (thus assuming independence between different time instants), we have that the trajectories followed by o compatible with the readings are: t_1: $L_1\,L_1\,L_0$, t_2: $L_1\,L_4\,L_0$, t_3: $L_4\,L_1\,L_0$, t_4: $L_4\,L_4\,L_0$, each with probability* $0.25 (= 0.5 \cdot 0.5 \cdot 1)$.

Thus, independence assumption allows for reasoning about trajectory probabilities as follows. Consider an object o moving over time interval $\mathcal{T} = [\tau_1..\tau_n]$, and the sequence of readings $\Theta = \langle \tau_1, R_1\rangle, \ldots, \langle \tau_n, R_n\rangle$, meaning that, at each $\tau_i \in \mathcal{T}$, o was detected by the set R_i of readers. Then, the probability that o followed the trajectory $t = l_1, \ldots, l_n$ (meaning that o was at location l_i at time point τ_i, for each $i \in [1..n]$) can be expressed (under the independence assumption) as: $p^a(t|\Theta) = \Pi_{i=1}^{n} p^a(l_i|R_i)$. Unfortunately, as highlighted in the following example, the probabilities returned by $p^a(t|\Theta)$ can be very different from those returned by the "actual" probability distribution $\Pr(t|\Theta)$.

EXAMPLE 2. *(continuing Example 1). Looking at the map, we can infer that t_1 is the only correct interpretation of the data, since L_0 and L_4 have no direct connection (as they are divided by a wall), and L_1 is directly connected (by means of a door) to L_0 but not to L_4. Thus, a correct probability distribution over the possible trajectories t_1, t_2, t_3, t_4 is as follows:* $\Pr(t_1|\Theta) = 1$, $\Pr(t_2|\Theta) = \Pr(t_3|\Theta) = \Pr(t_4|\Theta) = 0$, *where $\Theta = \theta_1, \theta_2$, and $\theta_1 = \langle \tau_1, \{r_1, r_5\}\rangle$, $\theta_2 = \langle \tau_2, \{r_0\}\rangle$.*

The point is that while $p^a(l|R)$ (and, in turn, $p^a(t|\Theta)$) is easy to obtain (as discussed above), finding a formulation for $\Pr(t|\Theta)$ is very hard, as it requires analyzing and encoding the correlations among possible positions over time.

In this paper, we address this problem: given a sequence of readings Θ and exploiting the knowledge of $p^a(l|R)$ (and thus $p^a(t|\Theta)$), how can we effectively and efficiently revise $p^a(t|\Theta)$ so that it takes into account possible correlations inside the data, thus obtaining a better estimate of $\Pr(t|\Theta)$? Intuitively enough, revising $p^a(t|\Theta)$ according to the known correlations can be viewed as a cleaning problem: the data to be cleaned are the (probabilistic) trajectories resulting from using $p^a(t|\Theta)$ to interpret the sequence of readings, and the cleaning task consists in revising the probabilities assigned to these trajectories.

Cleaning RFID data. Although a number of data-cleaning techniques for RFID data have been proposed (see Section 7), most of them do not exploit any knowledge on the map and on the motility characteristics (such as the maximum speed) of the monitored objects, even if the users who analyze the data are typically acquainted with these aspects. The point is that, from this knowledge of the domain, constraints can be naturally derived on the connectivity between pairs of locations (*direct unreachability* constraints) and/or on the time needed for reaching a location starting from another one (*traveling-time* constraints): as explained in the following example, these constraints [1] could be profitably used at least to discard interpretations of the data corresponding to inconsistent trajectories.

EXAMPLE 3. *Consider the scenario of examples 1 and 2. The map easily implies a set of* direct unreachability *constraints, one for each pair of rooms which are not directly connected through a door, such as L_0, L_4, and L_1, L_4. In particular, these two constraints are those exploited in Example 2 to infer that t_1 is the only possible trajectory in accordance with the readings and the map.*

The map implies further constraints, other than direct unreachability. For instance, it says that L_0 and L_5, although close to one another, are connected only by a pretty long path. Reasonably, it can be imposed that 15 *secs are required to go through this path (this will be called "*traveling-time *constraint"). This constraint implies that, when interpreting the readings for a person moving across the map, all the interpretations corresponding to trajectories where the person reached L_5 from L_0 in less than* 15 *secs should be discarded.* □

The discussion in Example 3 explains how considering integrity constraints can reduce uncertainty, as it allows trajectories to be removed from the valid interpretations of the data. Then, the point becomes how to reasonably combine the integrity constraints with the *a-priori* probabilistic model encoded by $p^a(l|R)$, in order to devise a mechanism for revising the a-priori probabilities of the remaining valid trajectories and making them sum up to 1.

A rigorous approach (commonly adopted in probabilistic databases to enforce constraints over probabilistic data [16, 7]) is to perform *conditioning*: starting from the a-priori probabilities (which do not take into account the constraints), the probabilities of the trajectories are re-evaluated as conditioned to the event that the constraints are satisfied. That is, given a set $\mathcal{IC}$ constraints, probabilities of the form $p^a(t|\Theta)$ are revised into $p^a(t|\Theta \wedge \mathcal{IC})$. This way, the probability of invalid trajectories becomes 0, while that of each valid trajectory becomes the ratio of its a-priori probability to the overall a-priori probability of the valid trajectories.

[1] It is worth noting that direct-unreachability (DU) and traveling-time (TT) constraints can be reasonably assumed to be available, as obtaining them does not require all that specific knowledge of the domain. In particular, DU constraints can be easily inferred from the map of the locations, and TT constraints can be easily obtained by reasoning on the distances between pairs of locations and the maximum speed v of the objects being monitored. Obviously, the map is known to any user asking for cleaned data (as she will use these data to analyze trajectories!), and reasonably assigning an upper bound on v is easy in several contexts, such as people visiting a museum or moving inside an office building. In fact, in our experiments, these constraints have been inferred automatically: the only input of the inference task were the map and the maximum speed v of the objects being monitored.

For instance, in the case discussed in examples 1, 2, 3, each a-priori probability $p^a(t_i|\Theta)$ is revised into $p^a(t_i|\Theta \wedge \mathcal{IC})$, where $p^a(t_2|\Theta \wedge \mathcal{IC}) = p^a(t_3|\Theta \wedge \mathcal{IC}) = p^a(t_4|\Theta \wedge \mathcal{IC}) = 0$, while $p^a(t_1|\Theta \wedge \mathcal{IC}) = \frac{0.25}{0.25} = 1$. In general, constraints reduce the number of valid trajectories, and the conditioning assigns to them "new" probabilities by keeping, for each pair of trajectories, the same probability ratios between their a-priori probabilities. For instance, consider 4 trajectories t_1, t_2, t_3, t_4 with a priori probabilities $p_1 = 0.5$, $p_2 = 0.25$, $p_3 = 0.2$, $p_4 = 0.05$, respectively. If t_3 and t_4 are invalid, then they will be discarded, while t_1 and t_2 will be assigned the (conditioned) probabilities $\frac{0.5}{0.75} = \frac{2}{3}$ and $\frac{0.25}{0.75} = \frac{1}{3}$. This reflects the fact that, before conditioning, t_1 was twice as probable as t_2.

Contribution. The revision problem of evaluating $p^a(t|\Theta \wedge \mathcal{IC})$ starting from $p^a(t|\Theta)$ is generally complex. The naive approach of enumerating the trajectories compatible with the readings, discarding those not satisfying the constraints, and finally revising the probabilities of the remaining ones is often infeasible, as the trajectories to deal with are too many. For instance, if, for each instant in the time interval $[1..100]$, two locations are compatible with the readings, we have to consider 2^{100} (that is about 10^{30}) trajectories.

Our main contribution is a framework which cleans RFID data by exploiting *direct unreachability* and *traveling-time* constraints (along with *latency* constraints, which will be introduced in the core of the paper). The proposed approach returns a compact representation (*ct-graph*) of the valid trajectories and their conditioned probabilities. This compact representation is obtained by an iterative algorithm which builds a graph whose nodes correspond to pairs ⟨*location*, *timestamp*⟩ and where paths from source to target nodes one-to-one correspond to the valid trajectories (source and target nodes refer to the first and last instant of the time interval of interest, respectively). This graph is built incrementally, aiming at preventing the creation of nodes and edges which would yield paths corresponding to invalid trajectories. The same algorithm assigns to each node or edge a probability obtained by suitably revising the a-priori probability of the corresponding pair ⟨*location*, *timestamp*⟩, so that the overall probability of a source-to-target path is the conditioned probability of the corresponding trajectory.

2. PRELIMINARIES

We consider a set $\mathcal{R} = \{r_1, \ldots, r_k\}$ of RFID readers, a single object o equipped with an RFID tag, and the set $\mathcal{L} = \{l_1, \ldots, l_n\}$ of locations among which o moves while monitored by the readers in $\mathcal{R}$. We assume that time is represented by the set of non-negative integers, and denote as $\mathcal{T} = [0..\tau_f]$ the time interval over which o is monitored by the readers in $\mathcal{R}$.

A reading θ of o is a pair $\langle \tau, R \rangle$, stating that, at time $\tau \in \mathcal{T}$, o was detected by all and only the readers in R, where $R \subseteq \mathcal{R}$ ($R{=}\emptyset$ means that o was detected by no reader at time τ). Its components τ and R are denoted as $\theta[\textit{time}]$ and $\theta[\textit{readers}]$.

A *reading sequence* (r-sequence) for o over $\mathcal{T}$ is a set Θ of readings of o containing, $\forall \tau \in \mathcal{T}$, a unique reading $\langle \tau, R \rangle$.

We assume given the probability distribution $p^a(l|R)$, defined over every $l \in \mathcal{L}$ and $R \subseteq \mathcal{R}$, representing the *a-priori* probability that an object is in the location l given that it has been detected by all and only the readers in R[2].

Given $p^a(l|R)$ and a reading θ, we can associate θ with the (discrete) random variable X_θ, which is defined over the locations in $\mathcal{L}$ and whose probability density function (PDF) is $f(X_\theta = l) = p^a(l \mid \theta[\textit{readers}])$. Basically, X_θ represents the alternative locations of the object at time $\theta[\textit{time}]$ which are compatible with the fact that, at that time, it was detected by the set $\theta[\textit{readers}]$ of readers. Each alternative location is assigned the probability implied by p^a.

Given $p^a(l|R)$ and an r-sequence Θ, we define the *(probabilistic) location sequence* (l-sequence for short) corresponding to Θ (according to $p^a(l|R)$) as the set $\Gamma = \{X_\theta \mid \theta \in \Theta\}$.

For the sake of presentation, we will represent any l-sequence by making explicit all the pairs ⟨*timestamp*, *location*⟩ compatible with the readings in the corresponding r-sequence. That is, the l-sequence Γ corresponding to the r-sequence Θ for o over $\mathcal{T}$ will be denoted as a pair $\Gamma = \langle \Lambda, p \rangle$, where:

- Λ is a set of pairs of the form $\lambda = \langle \tau, l \rangle$, with $\tau \in \mathcal{T}$ and $l \in \mathcal{L}$, containing at least one pair $\langle \tau, l \rangle$ for each $\tau \in \mathcal{T}$;
- p assigns to each pair $\langle \tau, l \rangle \in \Lambda$ the value $f(X_\theta = l)$, where θ is the reading at time τ. That is, p assigns to $\langle \tau, l \rangle$ the probability that the object was at location l at time τ, as implied by the PDF of the random variable corresponding to the reading at time τ.

Given an l-sequence $\Gamma = \langle \Lambda, p \rangle$, we assume that Λ contains only pairs which are assigned a non-zero probability by p.

From now on, in the examples we will consider the map in Fig 1(b).

EXAMPLE 4 (RUNNING EXAMPLE). *Consider the r-sequence* $\Theta = \{\langle 0, \{r_1\}\rangle, \langle 1, \{r_2\}\rangle, \langle 2, \{r_4\}\rangle\}$ *and* $p^a(l|R)$ *s.t.* $p^a(L_1|\{r_1\}) = \frac{6}{10}$, $p^a(L_2|\{r_1\}) = \frac{4}{10}$, $p^a(L_3|\{r_2\}) = \frac{1}{3}$, $p^a(L_3|\{r_4\}) = \frac{2}{3}$, $p^a(L_4|\{r_2\}) = \frac{2}{3}$, *and* $p^a(L_5|\{r_4\}) = \frac{1}{3}$. *The corresponding l-sequence* $\Gamma = \langle \Lambda, p \rangle$ *is s.t.* $\Lambda = \{\lambda_1 = \langle 0, L_1\rangle, \lambda_2 = \langle 0, L_2\rangle, \lambda_3 = \langle 1, L_3\rangle, \lambda_4 = \langle 1, L_4\rangle, \lambda_5 = \langle 2, L_3\rangle, \lambda_6 = \langle 2, L_5\rangle\}$, *and* $p(\lambda_1) = \frac{6}{10}$, $p(\lambda_2) = \frac{4}{10}$, $p(\lambda_3) = \frac{1}{3}$, $p(\lambda_4) = p(\lambda_5) = \frac{2}{3}$, *and* $p(\lambda_6) = \frac{1}{3}$. □

Given a pair $\lambda = \langle \tau, l \rangle \in \Lambda$, we denote with $\lambda[\textit{time}]$ and $\lambda[\textit{loc}]$ the first and the second component of λ, respectively.

DEFINITION 1 (TRAJECTORY). *Let* Θ *be an r-sequence over* $\mathcal{T}$ *and* $\Gamma = \langle \Lambda, p \rangle$ *be the l-sequence corresponding to* Θ. *A* trajectory *over* Γ *is a set* $t \subseteq \Lambda$ *of pairs such that, for each* $\tau \in \mathcal{T}$, *there is a unique pair* $\lambda \in t$ *such that* $\lambda[time] = \tau$. *The (a-priori) probability of* t *is* $p^a(t|\Theta) = \prod_{\lambda \in t} p(\lambda)$. □

For the sake of simplicity, in the following we assume given an r-sequence Θ, *thus we write* $p^a(t)$ *instead of* $p^a(t|\Theta)$.

The set of the trajectories over an l-sequence Γ is denoted as $\mathbf{T}(\Gamma)$. Given a trajectory t, the pair $\lambda \in t$ such that $\lambda[\textit{time}] = \tau$ is said to be the τ-th *step* of t.

Basically, a trajectory over an l-sequence Γ is obtained by picking, for each timestamp, one of the possible locations compatible with the reading at that timestamp, and thus represents a "possible interpretation" of the readings. Obviously, many trajectories are possible over the same l-sequence: in particular, their number is $\prod_{\tau \in \mathcal{T}} |\{\lambda \in \Lambda | \lambda[\textit{time}] = \tau\}|$, corresponding to all the ways of picking a location compatible with the observed readings and with $p^a(l|R)$ at each time point. Each of them is associated with a probability, implied by $p^a(l|R)$ under the assumption of independence between the random variables in Γ. This, in turn, means considering as independent the locations where the object was in any two time points. It is easy to see that $\sum_{t \in \mathbf{T}(\Gamma)} p^a(t) = 1$.

EXAMPLE 5. *Two out of the* 8 *trajectories over the l-sequence* $\Gamma = \langle \Lambda, p \rangle$ *of Example 4 are* $t_1 = \{\lambda_1, \lambda_3, \lambda_5\}$, *which means that object* o *went from* L_1 *to* L_3 *and stayed in* L_3 *for two consecutive timestamps, and* $t_2 = \{\lambda_1, \lambda_3, \lambda_6\}$, *which describes the case that*

[2]We assume that this probability does not depend on the detected object and is invariant over time. The extension of our framework to the more general case that this probability varies over time and type of objects is straightforward.

object o went from location L_1 to L_5 through L_3. Their a-priori probabilities are $p^a(t_1) = p(\lambda_1)\cdot p(\lambda_3)\cdot p(\lambda_5) = \frac{12}{90}$ and $p^a(t_2) = p(\lambda_1)\cdot p(\lambda_3)\cdot p(\lambda_6) = \frac{6}{90}$. □

In what follows, we will see how integrity constraints can limit the number of trajectories which should be considered as valid, thus reducing the uncertainty inherent to the readings.

3. CLEANING EXPLOITING INTEGRITY CONSTRAINTS

We consider three kinds of integrity constraints (namely, *direct unreachability*, *traveling time*, and *latency* constraints), whose definition is as follows.

Given two locations $l_1, l_2 \in \mathcal{L}$, a *direct unreachability* constraint (DU) has the form *unreachable*(l_1, l_2) and states that no object can reach l_2 from l_1 in one time point. Given two locations $l_1, l_2 \in \mathcal{L}$ and a non-negative integer ν, a *traveling time* (TT) constraint is of the form *travelingTime*(l_1, l_2, ν) and states that, for any object, the time needed to move from l_1 to l_2 is not less than ν. Finally, given a location $l \in \mathcal{L}$ and a non-negative integer δ, a *latency constraint* (LT) associated with l is denoted as *latency*(l, δ) and imposes that every time an object goes into location l, it must stay in l for at least δ time points.

Intuitively enough, DU constraints are implied by the structure of the map, and TT constraints are implied by the minimum distances between the locations and the maximum speed of the objects. We already discussed on how DU and TT are easy to be obtained, and on the fact that they can be reasonably assumed to be available in several contexts (see footnote 1 in the introduction). As regards latency constraints, they take into account the physical inertia of objects, as well as the processing times (for doing some job). Users can specify latency constraints even in the case that they do not encode a true knowledge of what happens in the real-world: they can be useful for discarding interpretations of the data corresponding to very short stays at some locations, which are not of interest for the data analysis which will be performed on the cleaning data. For instance, imposing *latency*("coffee room", 20s), removes from the interpretations of the data those stating that the monitored person stayed at room 1 for 2 minutes, then at the adjacent coffee room for 2 seconds, and then again at room 1 for some more minutes: a 2-second long stay at a coffee room can be considered too short to be considered as meaningful.

DEFINITION 2. *Let $\Gamma = \langle \Lambda, p \rangle$ be an l-sequence and $\mathcal{IC}$ a set of integrity constraints. A trajectory t over Γ is* valid *w.r.t. $\mathcal{IC}$ iff*

- *for each* latency$(l, \delta) \in \mathcal{IC}$*, it holds that, for each pair $\langle \tau, l \rangle$ in t such that either $\tau = 0$ or there is $\langle \tau - 1, l' \rangle \in t$, with $l' \neq l$, t contains all the pairs $\langle \tau + i, l \rangle$ with $i \in [1..\delta - 1]$;*
- *for each* unreachable$(l_1, l_2) \in \mathcal{IC}$*, there are no pairs $\langle \tau, l_1 \rangle$ and $\langle \tau + 1, l_2 \rangle$ in t; and*
- *for each* travelingTime$(l_1, l_2, \nu) \in \mathcal{IC}$*, there are no $\langle \tau_1, l_1 \rangle$ and $\langle \tau_2, l_2 \rangle$ in t, with $\tau_1 < \tau_2$, such that $\tau_2 - \tau_1 < \nu$.* □

The subset of $\mathbf{T}(\Gamma)$ containing all and only the trajectories which are valid w.r.t. $\mathcal{IC}$ will be denoted as $\mathbf{T}^{\models \mathcal{IC}}(\Gamma)$.

EXAMPLE 6. *Consider $\Gamma = \langle \Lambda, p \rangle$ from Example 4 and $\mathcal{IC} = \{$*latency$(L_4, 2)$, unreachable(L_2, L_3), travelingTime$(L_1, L_5, 3)\}$*, imposing that (i) if object o reaches location L_4, it must stay there for at least two consecutive timestamps; (ii) object o cannot directly reach location L_3 from location L_2; and (iii) object o cannot reach location L_5 from location L_1 in less than 3 timestamps.*

It is easy to see that trajectory t_1 of Example 5 is valid, as it does not violate any constraint in $\mathcal{IC}$. Trajectory t_2 in the same example is not valid, as it does not satisfy travelingTime$(L_1, L_5, 3)$*. Indeed, the difference between the timestamp of λ_6 and the timestamp of λ_1 is 2. It is easy to see that t_1 is the unique valid trajectory over Γ and $\mathcal{IC}$.* □

Given a set $\mathcal{IC}$ of integrity constraints and a location $l \in \mathcal{L}$, we define *maxTravelingTime*$(l) = \max\{\nu | $*travelingTime*$(l, l', \nu) \in \mathcal{IC}\}$, i.e., *maxTravelingTime*(l) is the maximum among the minimum traveling times required for object o to move from l to any other $l' \in \mathcal{L}$ according to the constraints in $\mathcal{IC}$.

In the rest of the paper, we assume that an l-sequence Γ and a set $\mathcal{IC}$ of integrity constraints are given.

3.1 Revising the probabilities of the trajectories: the problem

As explained above, integrity constraints can be exploited to clean the data, as they allow valid trajectories to be distinguished from invalid ones. In order to go through the cleaning process, the problem must be addressed of how to assign a reasonable probability to the valid trajectories, given that the a-priori probabilities of the trajectories do not take into account the cleaning effect of the constraints. In fact, invalid trajectories have non-zero a-priori probabilities (although these trajectories are not valid interpretations of the readings), and the a-priori probabilities of valid trajectories do not sum up to 1 (though these trajectories are the only possible interpretations of the readings).

A rigorous approach (commonly adopted in probabilistic databases to enforce constraints over probabilistic data [16, 7]) is to perform *conditioning*: starting from the a-priori probabilities, the probabilities of the trajectories are evaluated as conditioned to the event that the constraints are satisfied. That is, the probability of invalid trajectories becomes 0, while that of each valid trajectory becomes the ratio of its a-priori probability to the overall a priori probability of the valid trajectories. That is, given a trajectory $t \in \mathbf{T}(\Gamma)$, its probability $p^a(t)$ conditioned to the fact that the constraints in $\mathcal{IC}$ are satisfied is given by $p^a(t|\mathcal{IC})$[3], where:

- $p^a(t|\mathcal{IC}) = 0$ if t is not valid w.r.t $\mathcal{IC}$;
- $p^a(t|\mathcal{IC}) = \frac{p^a(t)}{\sum_{t' \in \mathbf{T}^{\models \mathcal{IC}}(\Gamma)} p^a(t')}$, otherwise.

Now, the probabilities of the valid trajectories sum up to 1, and the a-priori probabilities are taken into account as, for each pair of trajectories, the ratio between their conditioned probabilities is the same as that between their a-priori probabilities.

As discussed in the introduction, evaluating conditioned probabilities is, in general, a complex problem. Our approach is an ad-hoc solution for the considered scenario, and provides a compact representation of the valid trajectories and their conditioned probabilities. This compact representation is obtained by an iterative algorithm which, starting from an l-sequence, builds a graph (named *conditioned trajectory graph*) whose nodes correspond to pairs $\langle$*location, timestamp*$\rangle$ and where paths from source to target nodes one-to-one correspond to the valid trajectories (source and target nodes refer to the first and last instants in $\mathcal{T}$, respectively). The algorithm assigns probabilities to the source nodes and the edges of the conditioned graph, by suitably revising a-priori probabilities implied by $p^a(l|R)$, so that the overall probability of a source-to-target path (evaluated as the product of the revised probabilities of its source node and its edges) is the conditioned probability of the corresponding trajectory.

[3]We recall that this corresponds to $p^a(t|\Theta \wedge \mathcal{IC})$.

4. CONDITIONED TRAJECTORY GRAPHS

The *conditioned trajectory graph* (*ct-graph* for short) over an l-sequence $\Gamma = \langle \Lambda, p \rangle$ will be exploited to concisely represent trajectories over Γ in the presence of integrity constraints. The nodes of a ct-graph are said to be *location nodes*. Each location node n corresponds to a pair $\langle \tau, l \rangle \in \Lambda$ and is connected through directed edges to location nodes (chosen among the set of *successors* of n) corresponding to subsequent timestamps.

In the following, we formalize the concepts of location node (Section 4.1), successors of location nodes (Section 4.2), and finally provide the definition of ct-graph and formalize the concept of path over a ct-graph (Section 4.3).

4.1 Location nodes

A location node corresponds to a pair $\langle \tau, l \rangle \in \Lambda$. It stores some information summarizing the trajectory of the object until τ, and, in particular, pieces of information useful to check whether the paths including this node describe valid trajectories. This supplementary information is about the length of the current stay of the object o at l (which will be used to check latency constraints), and about some of the locations where o has been before τ (which will be used to check traveling-time constraints). More formally, given an l-sequence $\Gamma = \langle \Lambda, p \rangle$ for an object o and a set $\mathcal{IC}$ of integrity constraints, a location node n is a tuple of the form $\langle \tau, l, \delta, TL \rangle$, where $\langle \tau, l \rangle \in \Lambda$, $\delta \in \mathcal{T} \cup \{\perp\}$ (where $\perp$ means "*non-specified*"), and TL is a (possibly empty) set of pairs $\langle \tau_1, l_1 \rangle \in \Lambda$ (with $l_1 \neq l$), containing no two pairs coinciding in either the timestamp or the location. It represents the following facts:

A. o was in location l at time τ;
B. the stay of o at l started δ time points before τ;
C. for each $\langle \tau_1, l_1 \rangle \in TL$, the most recent detection of o at location l_1 (before τ) is at time point τ_1.

Given a location node $n = \langle \tau, l, \delta, TL \rangle$ and a trajectory t over Γ, we say that t *is compatible with* n if, according to t, the facts A, B, C hold.

For instance, both the trajectories $t_1 = \{\langle 0, L_1 \rangle, \langle 1, L_3 \rangle, \langle 2, L_3 \rangle\}$ and $t_2 = \{\langle 0, L_1 \rangle, \langle 1, L_3 \rangle, \langle 2, L_5 \rangle\}$ of Example 5 are compatible with location node $\langle 1, L_3, 0, \{\langle 0, L_1 \rangle\}\rangle$ since, according to both t_1 and t_2, o was in L_3 at timestamp 1, the duration of the stay of o at L_3 is 0, and the most recent detection of o at L_1 is at time point 0.

The information stored in a location node n can be used to state that some trajectories are invalid. Specifically, any $t \in \mathbf{T}(\Gamma)$ compatible with n is invalid if either i) or ii) hold:

i) All the following conditions are satisfied:
– $n.TL$ contains a pair $\langle \tau', l' \rangle$,
– there is *travelingTime*$(l', l'', \nu') \in \mathcal{IC}$ with $(n.\tau{+}1) - \tau' < \nu'$,
– the $(\tau{+}1)$-th step of t is at location l''.
In fact, t would represent that the object went from l' to l'' in less than ν' time points, thus violating the TT constraint;

ii) There is *latency*$(l, \delta') \in \mathcal{IC}$ with $n.\delta < \delta'$, and the $(\tau{+}1)$-th step of t is at a location different from l. In fact, t would represent that the object went away from l after staying less than δ' time points, thus violating the latency constraint.

For instance, consider location node $n = \langle 1, L_3, 0, \{\langle 0, L_1 \rangle\}\rangle$ and the TT constraint *travelingTime*$(L_1, L_5, 3)$. It is easy to see that the information stored in n can be used to state that any trajectory compatible with n and whose second step is at L_5 (such as trajectory t_2 of the running example) is invalid. Indeed, the facts that the second step of the trajectory is at L_5, and that the most recent detection of o in L_1 was at timestamp 0 (as stated in $n.TL$), mean that o would reach location L_5 from L_1 in less than 3 timestamps (thus violating the TT constraint).

Point i) means that an entry $\langle \tau', l' \rangle$ in $n.TL$ can be used only to check TT constraints involving location l', thus reporting it in $n.TL$ is useless if there is no TT constraint in $\mathcal{IC}$ involving l'. Analogously, point ii) means that the value of $n.\delta$ is useful only to check latency constraints defined over l, and thus reporting it is useless if $\mathcal{IC}$ contains no latency constraint of this kind. There are also other cases when reporting δ in n or some entry $\langle \tau', l' \rangle$ in $n.TL$ is useless for detecting invalid trajectories (according to the strategy in points i) and ii)). In particular, consider the case that $\mathcal{IC}$ contains a latency constraint $ic = latency(n.l, \delta^*)$, but evaluating δ according to its definition (point B.) yields $n.\delta > \delta^*$. This means that, for every trajectory compatible with n, the stay at location $n.l$ involving time point τ is long enough to satisfy ic, thus reporting δ is useless for discarding trajectories which, due to their $(\tau{+}1)$-th step, do not satisfy ic. Analogously, consider the case that there is some TT constraint involving l' as first argument, and that the most recent time point τ' when the object moved away from l' is such that $n.\tau - \tau' \geq$ *maxTravelingTime*(l'). Then, reporting $\langle \tau', l' \rangle$ in $n.TL$ is useless for discarding invalid trajectories based on their $(\tau{+}1)$-th step, since no choice of the location at the $(n.\tau{+}1)$-th step can violate some TT constraint involving l' as first argument.

Therefore, we will assume that in every location node n:

– $n.TL$ contains only entries of the form $\langle \tau', l' \rangle$, where location l' is involved in at least one TT constraint, and it holds that $n.\tau - \tau' <$ *maxTravelingTime*(l');
– $n.\delta$ is assigned a value in $\mathcal{T}$ iff there is a latency constraint *latency*$(n.l, \delta^*)$ and the value of $n.\delta$ (computed as specified in B) is such that $n.\delta < \delta^*$. Otherwise, $n.\delta$ is assigned $\perp$.

Under this assumption, we denote as $\mathcal{N}$ the set of the location nodes that can be defined over Γ and $\mathcal{IC}$, and as SN and TN the subsets of $\mathcal{N}$ of *source* and *target* nodes (i.e., locations nodes whose timestamps are the first and the last time points of $\mathcal{T}$), respectively. As it will be clearer later, when constructing the ct-graph, not all of the nodes in $\mathcal{N}$ will be materialized, but they are nevertheless used to formalize our approach.

EXAMPLE 7. *Continuing our running example, the set $\mathcal{N}$ over Γ and $\mathcal{IC}$ consists of the following location nodes:*

$$n_0 = \langle 0, L_1, \perp, \emptyset \rangle, \quad n_1 = \langle 0, L_2, \perp, \emptyset \rangle,$$
$$n_2 = \langle 1, L_3, \perp, \emptyset \rangle, \quad n_3 = \langle 1, L_3, \perp, \{\langle 0, L_1 \rangle\} \rangle,$$
$$n_4 = \langle 1, L_4, 0, \emptyset \rangle, \quad n_5 = \langle 1, L_4, 0, \{\langle 0, L_1 \rangle\} \rangle,$$
$$n_6 = \langle 2, L_3, \perp, \emptyset \rangle, \quad n_7 = \langle 2, L_3, \perp, \{\langle 0, L_1 \rangle\} \rangle,$$
$$n_8 = \langle 2, L_5, \perp, \emptyset \rangle, \quad n_9 = \langle 2, L_5, \perp, \{\langle 0, L_1 \rangle\} \rangle.$$

Location node n_2 means that: a) since $n_2.\tau = 1$ and $n_2.l = L_3$, object o was in L_3 at timestamp 1; b) since $n_2.\delta = \perp$, either there is no LT constraint over L_3, or its stay at L_3 started more than δ^* time points before 1, where latency(L_3, δ^*) *is the LT constraint over L_3; c) since $n_2.TL = \emptyset$, for every location l visited by the object in the past, either there is no TT constraints having l as first argument, or the object left l more than ν^* time points ago, where all the TT constraints* travelingTime$(l, l', \nu) \in \mathcal{IC}$ *having l as first argument are such that $\nu \leq \nu^*$.*

Analogously, n_5 means that: a) o was in L_4 at timestamp 1; b) its stay at L_4 started in the current timestamp; c) in the past, it stayed at L_1, from which it moved away at time point 0. □

4.2 Successors of a location node

We now introduce the concept of *successor* of a location node. Roughly speaking, location node $n_2 = \langle \tau + 1, l_2, \delta_2, TL_2 \rangle$ is a

successor of a location node $n_1 = \langle \tau, l_1, \delta_1, TL_1 \rangle$ if n_2 describes (in terms of facts A, B, C) a possible scenario at time $\tau + 1$ which is consistent with the integrity constraints and with the scenario at time τ described by n_1.

DEFINITION 3 (SUCCESSOR). *Given a pair of location nodes $n_1 = \langle \tau_1, l_1, \delta_1, TL_1 \rangle$ and $n_2 = \langle \tau_2, l_2, \delta_2, TL_2 \rangle$, n_2 is* successor *of n_1 iff the following conditions hold:*

1) $\tau_2 = \tau_1 + 1$*;*
2) unreachable$(l_1, l_2) \notin \mathcal{IC}$*;*
3) if $l_1 = l_2$*, then* $\delta_2 = \delta_1 + 1$[4]*;*
4) if $l_1 \neq l_2$ *and* latency$(l_1, \delta) \in \mathcal{IC}$*, then* $\delta_1 \geq \delta$ *(if* $\delta_1 \neq \perp$*) and either* $\delta_2 = 0$ *(if there is a latency constraint on* l_2 *in* $\mathcal{IC}$*) or* $\delta_2 = \perp$ *(otherwise);*
5) there is no pair $\langle \tau', l' \rangle \in TL_1$ *such that there is a constraint* travelingTime(l', l_2, ν) *in* $\mathcal{IC}$ *with* $\tau_2 - \tau' < \nu$*;*
6) $TL_2 = TL_1 \cup \{\langle \tau_1, l_1 \rangle \mid \text{travelingTime}(l_1, l, \delta) \in \mathcal{IC}\} \setminus \big(\{\langle \tau, l \rangle \in TL_1 \mid \tau_2 - \tau \geq \textit{maxTravelingTime}(l)\} \cup \{\langle \tau, l \rangle \in TL_1 \mid l = l_2\}\big)$ □

Basically, n_2 is a successor of n_1 iff 1) n_1 and n_2 refer to consecutive time points; 2) l_2 can be directly reached starting from l_1; 3) if at time τ_2 the object is still in $l_2 = l_1$, then this is consistently reflected by an increment of δ_2; 4) no stay constraint in $\mathcal{IC}$ involving l_1 is violated if the object moves from l_1 to another location l_2 at time τ_2; 5) no traveling time constraint in $\mathcal{IC}$ imposing that the time needed to move from a location belonging to TL_1 to l_2 is violated if the object moves from l_1 to another location l_2 at time τ_2; 6) TL_2 can be obtained by first augmenting TL_1 with the pair $\langle \tau_1, l_1 \rangle$ (this happens if there is a traveling time constraint involving l_1 as first argument), and then discarding those pairs of TL_1 which have become useless for checking TT constraints and those referring to a previous stay at l_2.

EXAMPLE 8. *Continuing our running example, it is easy to see that n_3 and n_5 are successors of n_0. Analogously, n_4 is a successor of n_1, and, finally, n_7 is a successor of n_3. The reader can easily check that there is no other pair of location nodes in $\mathcal{N}$ such that one is a successor of the other. For instance, n_2 is not a successor of n_1 since* unreachable$(L_2, L_3) \in \mathcal{IC}$*, thus Condition 2) of Definition 3 does not hold. Similarly, n_9 is not a successor of n_3 since* travelingTime$(L_1, L_5, 3) \in \mathcal{IC}$*, thus Condition 5) of Definition 3 does not hold. Finally, note that n_9 is not a successor of n_2 since both Condition 5) and Condition 6) of Definition 3 do not hold.* □

The following proposition states a property that will be fundamental for understanding how our cleaning algorithm exploits the notion of successor.

PROPOSITION 1. *If a non-target location node n admits no successor then every trajectory t compatible with n is not valid.*

4.3 ct-graphs and how to use them to encode valid trajectories

Based on the notions of location node, source and target nodes, and successor, the definition of ct-graph is as follows.

DEFINITION 4 (CT-GRAPH). *Let $\mathcal{N}$ be the set of location nodes over the l-sequence Γ, and $\mathcal{IC}$ a the set of integrity constraints. A conditioned trajectory graph (ct-graph) is a tuple $G = \langle N, E, p_N, \vec{p}_E \rangle$, where:*

1) $N \subseteq \mathcal{N}$*;*
2) $\langle N, E \rangle$ *is a graph (where N and E are the sets of nodes and edges, respectively) satisfying the following properties:*
   a) a pair $\langle n_1, n_2 \rangle$ belongs to E iff $n_1, n_2 \in N$ and n_2 is a successor of n_1;
   b) for every $n \in N$, there is at least one path from a source node to a target node in N which contains n;
3) $p_N : SN \rightarrow (0, 1]$ *is a PDF over the set of source nodes;*
4) $\vec{p}_E$ *is a set containing, for each non-target node n of G, a PDF p_E^n over its outgoing edges, that is, $\vec{p}_E = \{p_E^n | n \in N \setminus TN\}$ where $p_E^n : E_n \rightarrow (0, 1]$ and $E_n = \{\langle n, n' \rangle | \langle n, n' \rangle \in E\}$.* □

A path π over a ct-graph $G = \langle N, E, p_N, \vec{p}_E \rangle$ is a path from a source to a target node over the graph $\langle N, E \rangle$. It is easy to see that every path over G corresponds to a valid trajectory. Specifically, a path $\pi = n_0, \ldots, n_{\tau_f}$ over G corresponds to the valid trajectory $t = n_0[\lambda], \ldots, n_{\tau_f}[\lambda]$, where $n_i[\lambda]$ denotes the $\langle$*timestamp, location*$\rangle$ pair which n_i refers to. In fact, t is compatible with every location node along π, and, since the edges of G connect only pairs of nodes n_1, n_2 s.t. n_2 is successor of n_1, t must be valid w.r.t the integrity constraints.

What said above entails that a ct-graph can represent *only* valid trajectories (encoding them as paths of location nodes). In the next section, we present an algorithm for building a ct-graph that represents *all and only* the valid trajectories. Furthermore, our algorithm instantiates the functions p_N and $\vec{p}_E$ of G so that the probability $p(\pi)$ of a path π over G is equal to the conditioned probability $p^a(t|\mathcal{IC})$ of the corresponding trajectory t, where $p(\pi) = p_N(n_0) \times \prod_{i=0}^{\tau_f - 1} p_E^{n_i}(\langle n_i, n_{i+1} \rangle)$.

EXAMPLE 9. *Continuing our running example, it is easy to check that $G = \langle N, E, p_N, \vec{p}_E \rangle$, where $N = \{n_0, n_3, n_7\}$, $E = \{\langle n_0, n_3 \rangle, \langle n_3, n_7 \rangle\}$, $p_N(n_0) = 1$, and $p_E^{n_3}(\langle n_3, n_7 \rangle) = 1$ is a ct-graph (depicted in Fig. 7). As we will show in the next section, G is the ct-graph returned by our algorithm run over the l-sequence and the constraints of our running example. In fact, the (unique) path $\pi = n_0, n_3, n_7$ over G corresponds to the (unique) valid trajectory $\{\lambda_1, \lambda_3, \lambda_5\}$ of Example 6, and its probability is $p(\pi) = 1$.* □

5. THE CLEANING ALGORITHM

In this section we introduce Algorithm 1, which builds a conditioned trajectory graph G from an l-sequence Γ and a set $\mathcal{IC}$ of integrity constraints. In particular, the graph returned by our algorithm compactly represents all and only the valid trajectories over Γ, in the sense that each valid trajectory t corresponds to a path π over G, and vice versa. Moreover, it is such that the conditioned probability of each valid trajectory t coincides with that of the corresponding path π over G.

Algorithm 1 consists of two phases, called *forward* (lines 5–14) and *backward* phase (lines 15–29). The forward phase consists of one iteration for each timestamp $\tau \in \mathcal{T}$. At the first iteration ($\tau = 0$), the set N of the location nodes belonging to the ct-graph G being constructed is initialized with the set of the source nodes, and the probabilities of these nodes are set to those implied by the a-priori probabilities. Then, at τ-th iteration, N is progressively augmented with location nodes referring to the timestamp $\tau + 1$ and which are successors of some other location node already in N. Correspondingly, the edges between location nodes referring to timestamp τ and their successors are added to E, and their probabilities are set to those implied by the a-priori probability function. The fact that a node n is added to N at τ-th iteration means that there is a trajectory t compatible with n which is valid if only its

[4]We override operator $+$ so that $\perp + 1 = \perp$, and $x + 1 = \perp$, when x is equal to the duration of the latency constraint over l_1.

first $\tau+1$ steps are considered. Clearly, in a subsequent iteration, such a trajectory t could be recognized as invalid due to the fact that n, or any of the other nodes with which t is compatible, admits no successor (see Proposition 1). This makes it possible the presence, at the end of the forward phase, of some "leaf" nodes (such as n mentioned above) that are non-target nodes having no successor. The backward phase deals with removing these leaf nodes, along with all of the other nodes which become leaves in this phase due to the deletion of all its successors. Furthermore, during this phase, the probabilities of edges and those of source nodes are conditioned, that is they are revised to take into account the probabilities of the trajectories that are recognized as invalid.

We now provide a detailed description of Algorithm 1, showing how it works over our running example.

While building a ct-graph, Algorithm 1 exploits an auxiliary variable $n.loss$ for each node n of the graph. Roughly speaking, at each step, the value of $n.loss$ is the ratio between the overall probability of the trajectories compatible with n which have been recognized as invalid, and the overall probability of the trajectories compatible with n. Moreover, Algorithm 1 exploits also a priority queue Q which is used to store (in descending timestamp order) the location nodes n for which n.loss is greater than 0, which are the location nodes that have to be processed during the backward phase.

The initialization phase (lines 1-4) of Algorithm 1 is as follows. First of all, the set N of the nodes of the graph G being constructed is initialized with the set of the source nodes yielded by function *sourceNodes* that builds them from the l-sequence Γ (line 2). Next, the probability $p_N(n)$ of each source node n is set to the probability of the pair $\langle 0, l\rangle$ of Γ which n refers to (line 3), and the queue Q is set to be empty.

EXAMPLE 10. *Consider the* l-sequence $\Gamma = \langle \Lambda, p\rangle$ *and the set $\mathcal{IC}$ of constraints of our running example. In the initialization phase, Algorithm 1 works as follows. At line 2, function sourceNodes builds the source location nodes n_0 and n_1 (see Example 7), and puts them into N. Next, at line 3, $p_N(n_0)$ and $p_N(n_1)$ are assigned probabilities $p(\lambda_1)=\frac{6}{10}$ and $p(\lambda_2)=\frac{4}{10}$, respectively.* □

Algorithm 1 Building the conditioned trajectory graph

Require: $\Gamma = \langle \Lambda, p\rangle, \mathcal{IC}$
Ensure: $G = \langle N, E, p_N, \vec{p}_E\rangle$ over Γ and $\mathcal{IC}$
1: $SN \leftarrow$ *sourceNodes*(Γ)
2: $N \leftarrow SN$
3: $\forall\, n = \langle 0, l, \cdot, \cdot\rangle \in N$, $p_N(n) \leftarrow p(\langle 0, l\rangle)$
4: $Q \leftarrow \emptyset$
5: **for all** $\tau \in [0..\tau_f - 1]$ **do**
6: **for all** $n \in N$ s.t. $n[\lambda][time] = \tau$ **do**
7: $\mathcal{S} \leftarrow buildSuccessors(n, \Gamma, \mathcal{IC})$
8: **for all** $n' \in \mathcal{S}$ **do**
9: let $n' = \langle \tau+1, l', \delta', TL'\rangle$
10: $N \leftarrow N \cup \{n'\}$, $E \leftarrow E \cup \{\langle n, n'\rangle\}$
11: $p_E^n(\langle n, n'\rangle) \leftarrow p(\langle \tau+1, l'\rangle)$
12: $n.loss = 1 - \sum_{\langle n,n'\rangle \in E} p_E^n(\langle n, n'\rangle)$
13: **if** $n.loss > 0$ **then**
14: *in*(Q, n)
15: **while** Q is not empty **do**
16: $n \leftarrow$ *out*(Q)
17: **if** $n.loss < 1$ **then**
18: **for all** $\langle n, n'\rangle \in E$ **do**
19: $p_E^n(\langle n, n'\rangle) \leftarrow \frac{p_E^n(\langle n,n'\rangle)}{(1-n.loss)}$
20: **for all** $\langle n', n\rangle \in E$ **do**
21: $old \leftarrow p_E^{n'}(\langle n', n\rangle)$
22: $p_E^{n'}(\langle n', n\rangle) \leftarrow p_E^{n'}(\langle n', n\rangle) - n.loss \times old$
23: $n'.loss \leftarrow n'.loss + n.loss \times old$
24: **if** $n' \notin Q$ **then**
25: *in*(Q, n')
26: **if** $n.loss = 1$ **then**
27: $E \leftarrow E - \{\langle n', n\rangle\}$
28: **if** $n.loss = 1$ **then**
29: $N \leftarrow N - \{n\}$
30: **for all** $n \in (N \cap SN)$ **do**
31: $p_N(n) \leftarrow \frac{p_N(n)}{\sum_{n' \in (N \cap SN)} p_N(n')}$
32: **return** G consisting of $\langle N, E, p_N, \vec{p}_E\rangle$

Forward phase

In the forward phase, for each timestamp τ, for each location node n referring to τ and belonging to N, the set $\mathcal{S}$ of the location nodes n' such that n' is a successor of n is built by function $buildSuccessors$ (line 7). Next, each node n' of $\mathcal{S}$ is added to N and consequently the edges of the form $\langle n, n'\rangle$ are added to the set E of G (line 10). Next, being $n' = \langle \tau+1, l', \delta', TL'\rangle$ a successor of n, the probability $p_E^n(\langle n, n'\rangle)$ of the edge $\langle n, n'\rangle$ is set equal to the probability of the pair $\langle \tau+1, l'\rangle$ provided by function p in Γ. Once all the successors of n have been added to G, variable $n.loss$ is set to the one's complement of the sum of the probabilities of the the outgoing edges of n (that is, the sum of the probabilities of the pairs $\langle \tau+1, l''\rangle$ which do not appear in any of the successors of n) (line 12). Finally, n is added to the queue Q if $n.loss$ is greater than zero (line 14).

EXAMPLE 11. *After the initialization phase, Algorithm 1 works as follows. At iteration $\tau = 0$ (line 5), both n_0 and n_1 are processed. In the case $n = n_0$, the inner loop processes the location nodes n_3 and n_5, which, as shown in Example 8, are the only successors of n_0. Both nodes n_3 and n_5 are added to N, and both the edges $\langle n_0, n_3\rangle$ and $\langle n_0, n_5\rangle$ are added to E, with probabilities $p(\langle n_0, n_3\rangle) = p(\langle 1, L_3\rangle) = 1/3$ and $p(\langle n_0, n_5\rangle) = p(\langle 1, L_4\rangle) = 2/3$. At the end of inner loop, $n_0.loss$ is set to 0 (line 12) and thus n_0 is not added to Q. It is worth noting that location nodes that are not successors of any node in the graph are disregarded. For instance, though n_2 refers to timestamp 1 (see Example 7), it is disregarded, as it is not a successor of n_0 or of n_1 (as said in Example 8). In the case $n = n_1$, the inner loop only processes the location node n_4, which is the only successor of n_1 (see Example 8). Thus, n_4 is added to N, the edge $\langle n_1, n_4\rangle$ is added to E, and the probability $p(\langle n_1, n_4\rangle)$ is set equal to $p(\langle 1, L_4\rangle) = 2/3$. Variable $n_1.loss$ is set to $1 - 2/3 = 1/3$, and n_1 is added to Q (line14). The structure of G at end of the iteration $\tau = 0$ of the outermost loop is depicted in Fig. 2.*

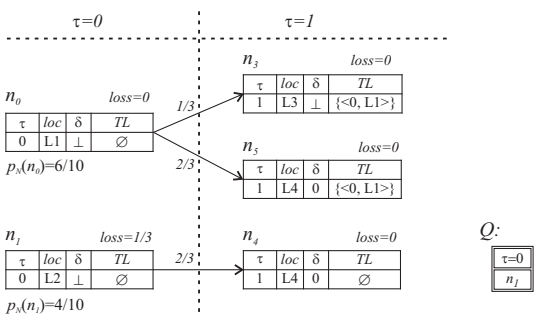

Figure 2: G and Q at the end of iteration $\tau = 0$

At iteration $\tau = 1$ (line 5), n_3, n_5 and n_4 are processed. In the case $n = n_3$, the inner loop only processes the location node n_7 which is the only successor of n_3 (as said in Example 8). Thus, n_7 is added to N, the edge $\langle n_3, n_7\rangle$ is added to E, and the probability $p(\langle n_3, n_7\rangle)$ of the edge is assigned with $p(\langle 2, L_3\rangle) = 2/3$. Next, $n_3.loss$ is assigned with $1 - 2/3 = 1/3$ and n_3 is added to Q. In both the cases $n = n_5$ and $n = n_4$, since there is no location node which is a successor of n_5 or n_4 (see Example 8), $n_5.loss$ and $n_4.loss$ are both assigned with 1 and n_5 and n_4 are both added to Q. The structure of G at end of the forward phase is shown in Fig. 3. □

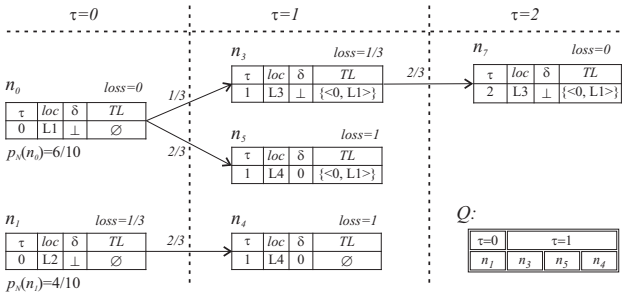

Figure 3: G and Q at the end of the forward phase.

Backward phase

During the backward phase, each location node which has been added to the priority queue Q is processed. During this phase, the probability values attached to the edges (and to source nodes), as well as the values of the *loss* variables associated with location nodes, are revised. In particular, at line 16, a node n among those having the highest timestamp is extracted from Q. Then, if $n.loss$ is lower than 1 (meaning that n has at least one outgoing edge), the probability of each of the outgoing edges of n is conditioned, that is, its current value is divided by the sum of the probabilities of the outgoing edges of n (which is equal to $1 - n.loss$) (line 19). Next, in order to propagate backward the value of variable $n.loss$, the probability value each ingoing edge $\langle n', n\rangle$ of n, along with the value of variable $n'.loss$ of each node n' of which n is successor is revised. Specifically, the probability of each ingoing edge $\langle n', n\rangle$ of n is decremented by the value of $n.loss$ multiplied by the old probability value of the edge $\langle n', n\rangle$ (line 22). Correspondingly, the value of $n'.loss$ of each node n' of which n is successor is incremented by the value of $n.loss$ multiplied by the old probability value of the edge $\langle n', n\rangle$ (line 23). Next, since $n'.loss$ has been incremented at line 23 and thus it is greater than 0, node n' is added to Q, if it is not already present in it (line 25). Furthermore, in the case $n.loss = 1$, the edge $\langle n', n\rangle$ is removed from E (line 27).

At the end of the loop scanning the ingoing edges of n, if $n.loss = 1$ (i.e., n is a leaf node) n is removed from N (line 29). Finally, the probability $p_N(n)$ of each source node n in N is conditioned, that is $p_N(n)$ is divided by the sum of the probabilities of the source nodes belonging to N (line 31).

EXAMPLE 12. *Continuing our example, the backward phase of Algorithm 1 is as follows. At the first iteration of the* while *loop (line 15), node n_4 is removed from Q. Since $n_4.loss$ is equal to 1, lines 17–19, which perform the conditioning of the outgoing edges of the current node, are skipped. Thus, Algorithm 1 processes the ingoing edges of n_4 (lines 20–27). Specifically, as n_4 has only the incoming edge $\langle n_1, n_4\rangle$, the probability value of $\langle n_1, n_4\rangle$ is set to $2/3 - 2/3 = 0$, and, correspondingly variable $n_1.loss$ is set to $1/3 + 2/3 = 1$. Next, at line 25, no operation is performed since n_1 is already present in Q. Then, at line 27, the edge $\langle n_1, n_4\rangle$ is removed from G, and, at line 29, n_4 is removed from N. The structure of G at end of the while loop processing n_4 is shown in Fig. 4.*

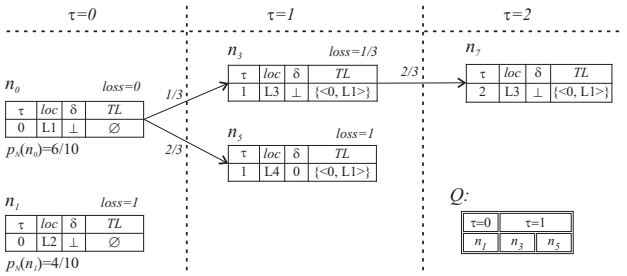

Figure 4: Structure of G after removing n_4 from Q and processing it.

The second iteration of the while *loop processes the node n_5. Since $n_5.loss$ is equal to 1, no conditioning of outgoing edges is performed. Then, Algorithm 1 processes the ingoing edges of n_5 (lines 20–27), and, reasoning as in the case described above, the probability value $\langle n_0, n_5\rangle$ is set to 0, and variable $n_0.loss$ is set to $2/3$. Next, n_0 is inserted into Q (line 25), the edge $\langle n_0, n_5\rangle$ is removed from G (line 27), and n_5 is removed from N (line 29). The structure of G at end of the while loop processing n_5 is shown in Fig. 5.*

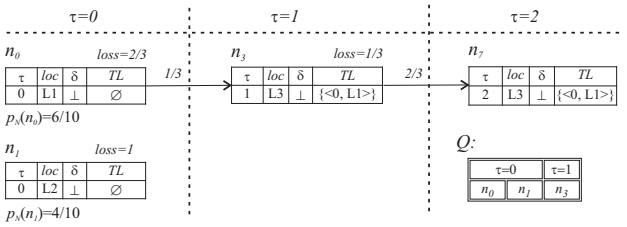

Figure 5: Structure of G after removing n_5 from Q and processing it.

At the third iteration of the while, *node n_3 is removed from Q. Since $n_3.loss$ is lower than 1, Algorithm 1 performs the conditioning of the outgoing edges of n_3 so that the probability of the unique outgoing edge $\langle n_3, n_7\rangle$ of n_3 is set to 1. Next, the unique ingoing edge $\langle n_0, n_3\rangle$ of n_3 is scanned, the probability of $\langle n_0, n_3\rangle$ is set to $1/3 - 1/3 \cdot 1/3 = 2/9$, and variable $n_0.loss$ is set to $2/3 + 1/3 \cdot 1/3 = 7/9$. Since n_0 is already in Q, the iteration processing n_3 ends without doing anything else. The graph resulting from this is shown in Fig. 6.*

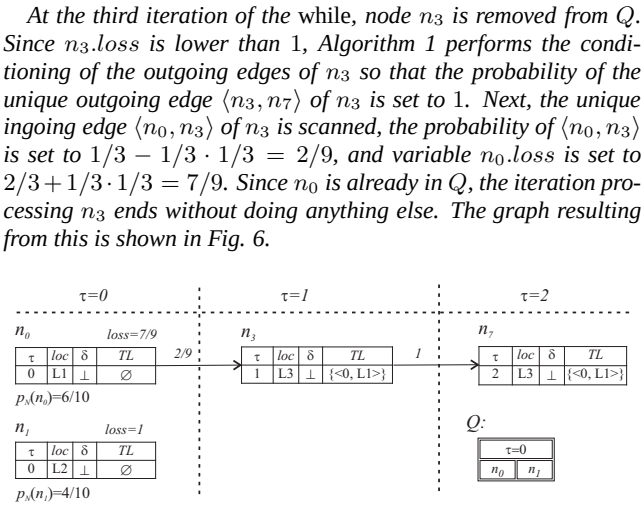

Figure 6: Structure of G after removing n_3 from Q and processing it.

At the fourth iteration of the while *loop the node n_1 is is removed from Q and processed. Since $n_1.loss$ is equal to 1 and it has no ingoing edges, the only operation performed is that of removing n_1 from N (line 29). Finally, the node n_0 is removed from Q and processed. The conditioning of its outgoing edges (lines 17–19) entails that the probability of $\langle n_0, n_3\rangle$ is set to 1. As n_0 has no ingoing edges and $n_0.loss < 1$, the lines 20–29 are skipped. At this stage, the probability of the source node n_0 (the unique that is still in N) is conditioned, obtaining $p_N(n_0) = 1$, and the algorithm returns the ct-graph shown in Fig. 7.* □

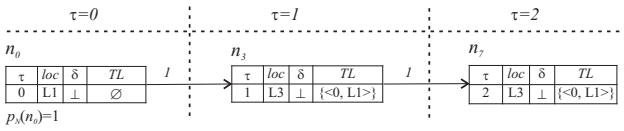

Figure 7: Ct-graph returned by Algorithm 1.

As regards the computational complexity of our algorithm, it is easy to see that Algorithm 1 works in polynomial time w.r.t. the length of trajectories over the l-sequence Γ (that is, the size of $\mathcal{T}$)[5]. The actual computational cost of our algorithm will be experimentally validated in the next section.

Remark: querying cleaned data. We do not give details on how queries can be evaluated on ct-graphs, as ct-graphs can be seen as Markovian streams and thus warehousing systems for Markovian streams, such as Lahar [22, 18, 19], can be used to store and query ct-graphs. Indeed, in [22, 18, 19] a Markovian stream with only one node (tuple) for each pair $\langle \tau, l\rangle$ is used for encoding RFID data, meaning that each stay has no memory of what happened in the past stays. However, the Lahar data model is expressive enough to allow storing more than one tuple for each pair $\langle \tau, l\rangle$, thus allowing the representation of ct-graphs. In other words, our ct-graphs can be viewed as a way of simulating memory in Markovian streams, by creating different nodes for a stay, depending on its alternative x-long sequences of past stays (where x depends on the constraints).

6. EXPERIMENTAL VALIDATION

All the experiments have been carried out on an Intel i7 CPU with 8GB RAM running Windows 7.

6.1 Data Sets

We considered two synthetic data sets (namely, SYN1 and SYN2), obtained using our synthetic data generator on two buildings of four and eight floors respectively. An example map of a floor is reported in Fig. 1(a). Each synthetic data set consists of 100 trajectories, that is, 25 trajectories for each duration in $\{10m, 60m, 90m, 120m\}$. The 25 trajectories in SYN1 with duration x minutes will be denoted as SYN1$_x$ (the same for SYN2).

6.2 The a-priori probability distribution $p^a(l|R)$

As discussed in the introduction, several ways can be adopted for obtaining $p^a(l|R)$. In our experiments, $p^a(l|R)$ was obtained as follows. First, the map in Fig. 1(a) was partitioned according to a regular grid of square cells having size 0.5mt×0.5mt. Then, a tag was kept inside each of these cells for 30 seconds and the bi-dimensional array F (consisting of one row for each reader and one column for each cell) was progressively filled by reporting in each cell $F[r, c]$ the number of times that the tag was detected by reader r during its 30-second long stay inside cell c.

After populating F, $p^a(l|R)$ was obtained as follows:

$$p^a(l|R) = \begin{cases} \frac{1}{|\mathcal{L}|}, \text{ if } \forall c \in \mathit{Cells} \prod_{r\in R} F[r,c] = 0; \\ \frac{\sum_{c\in \mathit{Cells}(l)} \prod_{r\in R} F[r,c]}{\sum_{c\in \mathit{Cells}} \prod_{r\in R} F[r,c]}, \text{ otherwise} \end{cases}$$

where *Cells*(l) and *Cells* represent the cells inside location l and all the cells inside the map, respectively.

Condition $\forall c \in \mathit{Cells} \prod_{r\in R} F[r, c] = 0$ means that there is no cell c such that the tag used for learning F has been detected by all the readers in R when it was in c. In this case, we have no a-priori knowledge about the probability that an object is in a location given that it has been detected by the readers in R, thus we assume a uniform distribution over $\mathcal{L}$.

6.3 Integrity constraints

Over each data set, the following sets of constraints were considered:

- DU: it contains all the DU constraints implied by the map;
- LT: it contains an LT constraint for every location but the corridors, imposing that the duration of every stay at any location must be not less than 5 seconds.
- TT: it contains a TT constraint for every pair of locations which are connected, but not directly connected. For each pair of locations L_1, L_2 of this kind, the constraint was automatically generated by taking the ratio between the minimum walking distance between L_1 and L_2, and the maximum speed of a person walking inside a building (which was assumed to be 2 ms^{-1}).

6.4 Synthetic data generator

The data generator consists of two modules: the trajectory generator and the reading generator.

The former takes as input the number num and the duration T_f of the trajectories to be generated, and a graph of locations, whose nodes represent rooms (described by the coordinates of their top-left and bottom-right corner) and whose edges represent doors between rooms (edges are labeled with the coordinates of the corresponding doors). The num generated trajectories consist of one triple $\langle x, y, \tau\rangle$ for each $\tau \in [0..T_f]$, where x, y are coordinates inside the space covered by the map.

Each trajectory is constructed iteratively, and, at each step, the following trajectory portion is generated. First, the object (denoted as o) moves (with velocity v) from an "entrance point" ep of the current room l to a "rest-point" rp inside l; then, o stays at rp for *lat* time instants; finally, o moves (at velocity v) to an "exit point" ep' of l. The rest-point rp is randomly generated inside the portion of space covered by the room, the latency time *lat* is randomly generated in $[30s..60s]$, the velocity v is randomly generated in $[1ms^1..2ms^{-1}]$, while the exit-point ep' is randomly generated among the doors connecting l with other rooms. The choice of ep' determines the room and the entrance point at the next step (at the first step, the current location and its entrance point are randomly generated). The generation of a trajectory ends when the input duration has been reached.

The reading generator takes as input a trajectory generated by the first module, a grid-partitioning of the map, and a model for the reading capacity of the RFID antennas, in terms of an array $F[r, c]$ analogous to that defined in the previous section. Specifically, each cell in $F[r, c]$ represents the percentage of times that an object staying for consecutive time points inside the cell c of the grid is detected by reader r.

[5]We refer to data complexity, according to which the size of the set of integrity constraints and the size of the set of locations are fixed.

The readings are generated by transforming each triplet $\langle x, y, \tau \rangle$ into a reading $\langle R, \tau \rangle$, where R is obtained as follows. First, the cell c of the grid containing the point with coordinates x, y is determined. Then, for each reader r, a number z is randomly generated in $[0, 1]$, and r is put into R if and only if z is less than $F[r, c]$. This means interpreting $F[r, c]$ as the probability that an object in c is detected by r, and assuming that readers behave independently.

6.5 Data Cleaning Cost

Figures $8(a)$ and $8(b)$ report the average running time of CTG over SYN1 and SYN2 vs. trajectory length. In all the diagrams, CTG(X) denotes our approach considering all the constraints in X.

The curves show that:

- for the same data set and set of constraints, the running time increases linearly with the trajectory length.
- for the same data set and duration of trajectories, considering a wider set of constraints slows down the cleaning task. This was rather expected, as exploiting LT and TT yields a larger number of nodes in the conditioned trajectory graph.
- CTG runs faster on SYN1 than on SYN2, especially when also TT are considered. In fact, the larger the map, the longer the maximum duration of the generated TT constraints over each location. This may increase the number of location nodes which must be created over the same pair $\langle$*time point, location*$\rangle$.

6.6 Accuracy of query answers over cleaned data

In this section we analyze the accuracy of the query answers evaluated over the cleaned data. We considered two kinds of queries (namely *stay* and *trajectory* queries), and evaluated them over the ct-graphs returned by our approach.

A stay query asks where the monitored object was at a specified time point, while a trajectory query asks whether the trajectory followed by the monitored object matches a pattern. Specifically, a pattern is a sequence of location conditions, where a location condition is either i) a location name, possibly followed by $[n]$, where n is a number of time points, or ii) the wildcard symbol "?".

The location conditions in a pattern must be expanded as follows:

- "?" $\rightarrow$ a (possibly empty) sequence of (any) locations;
- $l \rightarrow$ a sequence of l of length at least one;
- $l[n] \rightarrow$ a sequence of l of length at least n.

The answer to a trajectory query is *yes* iff the sequence of locations travelled by the object can be obtained by expanding the location conditions in the query pattern. For instance, trajectory query $q = ?l_1[3]?l_2[2]?$ asks whether the object, at some point, stayed at l_1 for at least 3 consecutive time points, and then travelled towards l_2 (passing by any other locations), where it stayed for at least two consecutive time points.

The probabilistic answers of stay and trajectory queries over ct-graphs are the natural probabilistic extensions of their deterministic semantics. Thus, given a ct-graph G, the answer $q(G)$ of a stay query q over time point τ is a set of locations, each associated with the probability that the object was at the location at time τ according to G (this means assigning to a location the sum of the probabilities of the trajectories represented in G whose τ-th step is at this location). Similarly, given a ct-graph G, the answer $q(G)$ of a trajectory query q is *yes* (resp., *no*) with probability p (resp., $1 - p$), where p is the sum of the probabilities of source-to-target paths over G representing trajectories matching the query pattern.

Correspondingly, given a stay query q and a ct-graph G, the accuracy of the answer $q(G)$ will be measured as the probability associated with L in $q(G)$, where L is the answer of q evaluated on the actual trajectory[6]. Similarly, given a trajectory query q and a ct-graph G, the accuracy of the answer $q(G)$ will be measured as the probability associated with the same answer (*yes* or *no*) returned by evaluating q on the actual trajectory.

As regards stay queries, for all the considered data sets, we considered a query workload consisting of 100 stay queries over each trajectory. Each query was generated by randomly picking a time point of the trajectory. The average accuracies of the answers of stay queries for the two data sets are reported in Figure $9(a)$.

As regards trajectory queries, we randomly generated 50 queries over each trajectory, whose pattern contain two, three or four location symbols, separated by symbol ?. Specifically, each trajectory query is generated as follows. First, a number x is randomly chosen in $2, 3, 4$. Then, x locations $l_1, \ldots, l_x$ are randomly chosen among those appearing in the map, and, for each picked location l_i, a number n_i in $-1, 3, 5, 7, 9$ is generated. The generated pattern is [7] $?\, l_1[n_1]\, ? \ldots ?l_x[n_x]\, ?$.

Fig. $9(b)$ and Fig. $9(c)$ report the average accuracy of trajectory queries vs. the two data sets (b) and the query length (for the case of SYN2) (c), respectively.

6.7 Querying efficiency and ct-graph size

The average query execution times of CTG over the two data sets vs. the trajectory length are reported in Fig. $8(c)$. The query execution times of CTG grows linearly with trajectory length. However, when running queries over ct-graphs obtained using only DU and LT constraints much faster execution times are obtained: this derives from the fact that these ct-graphs are smaller than those obtained when considering also TT constraints.

As regards the size of ct-graphs, we obtained that the average memory needed to store ct-graphs representing 120min-long trajectories is 25 Mb in the case that DU, LT, TT constraints are used, and it is only 640 kb in the case that DU constrains are used by CTG.

7. RELATED WORK

The management of RFID data has been studied from different perspectives. The definition of models for suitably representing RFID data has been addressed in [1, 3, 17], where several technological aspects and management issues for RFID data have been discussed, and a number of requirements in data modeling and software management have been highlighted. In the same data-modeling context, the problem of defining an efficient warehousing model along with techniques for summarizing and indexing RFID data has been investigated in [9, 8]. These approaches can be viewed as lossless compression techniques for RFID data. Lossy compression techniques for RFID-data are instead proposed in [6, 5] and in [2], where compression can be also seen as a form of cleaning.

The problem of cleaning RFID data was more specifically addressed in several other works. In [14], cleaning technique *SMURF* was proposed, specifically designed for dealing with false negatives. SMURF is a combination of sampling techniques with an adaptive, declarative smoothing filter, which determines whether an object which has not been detected by a reader was actually in the

[6] For SYN1 and SYN2, the actual trajectories were generated by the trajectory generator module.

[7] In the case that $n_i = -1$, then condition l_i is used instead of $l_i[n_i]$.

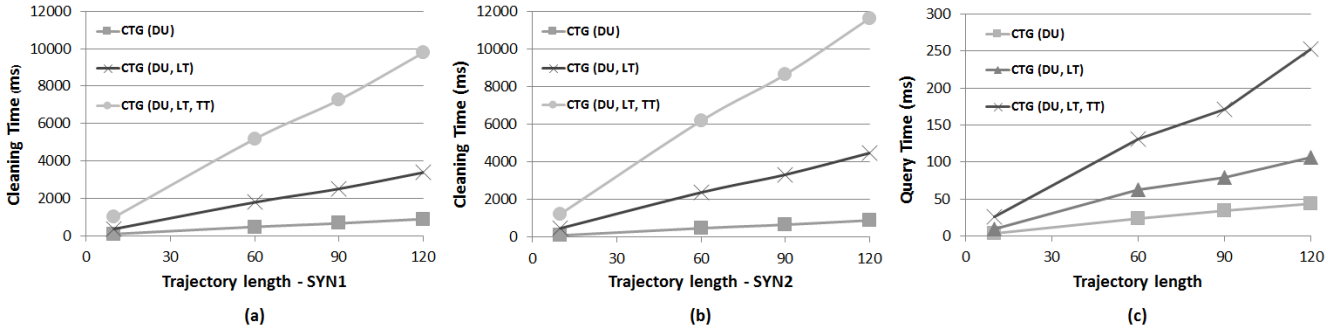

Figure 8: Average cleaning times vs SYN-1 (a) and SYN-2 (b) and average query time on SYN-1/SYN-2

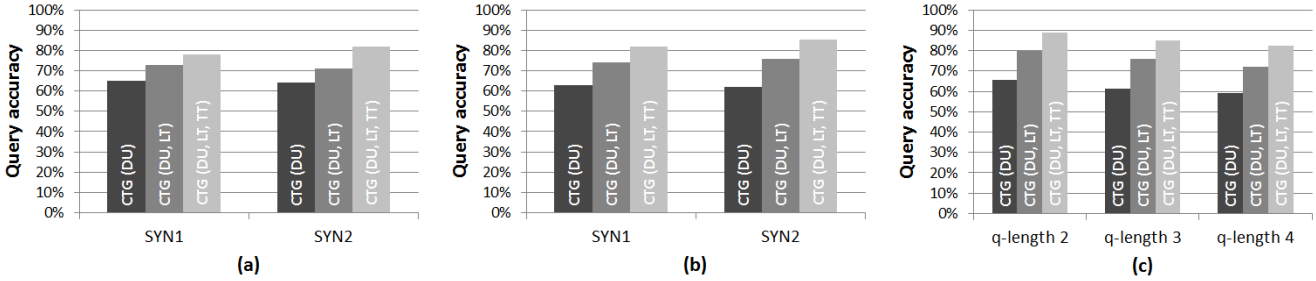

Figure 9: Average accuracy for stay queries (a) and trajectory queries (b) vs. the two datasets, and for trajectory queries over SYN2 vs. the query length (c)

query range by looking at the "history" of detections of the same object at the same location. SMURF works at level of readers and not of locations, and cleans the sequences of readings generated by distinct readers by considering them separately. Thus, differently from our approach, it can not exploit the spatio-temporal correlations described by the constraints considered in our scenario when cleaning the data.

In [4, 25], sampling techniques guided by the constraints are used to clean RFID data, as they generate (weighted) samples satisfying the constraints (the samples can be viewed as cleaned data in the sense that they are representative of consistent interpretations of the data). Although "sampling under constraints" is a general statistical framework which, in principle, may deal with any kind of constraint, these works use this framework under constraints involving the positions where the same object is detected at a given time point (they do not explicitly deal with constraints involving the positions at different time points). The main relationship between our work and sampling techniques is that our ct-graph can be used a basis for efficiently performing sampling: any trajectory picked from the ct-graph is valid, thus sampling can be done over it with no need to devise mechanisms for avoiding the generation of invalid samples. In future work, we plan to investigate differences in terms of efficiency between constructing a ct-graph and picking samples from it w.r.t. some adaption of the "sampling under constraints" framework to RFID-trajectory data.

In [23], the cleaning problem is addressed in a different scenario: the reader is mobile, its position is known at each time point (possibly with some approximation), and the problem is that of determining the position (in terms of spatial coordinates) of tagged objects from the noisy and incomplete stream generated by the reader. In [10], a general cleaning framework is introduced, which relies on a collection of cleaning methods, associated with costs. The application of the framework on an RFID data set results in a cleaning plan that optimizes the overall accuracy-adjusted cleaning costs by determining the conditions under which inexpensive methods are appropriate, and those under which more expensive methods are necessary. In [24], some form of cleaning (elimination of duplicates) is performed while suitably populating a database instance where the represented events are the changes of locations of the monitored objects. [15] is a general probabilistic framework for fixing streaming data (thus, also RFID data) which are inconsistent w.r.t. some integrity constraints. The fixing strategy consists in adding probabilistic tuples whose probabilities are determined so that the integrity constraints are satisfied. The constraints are inequalities on the number of tuples satisfying a property X, and are considered to be satisfied when the *average* number of tuples having property X satisfies the inequality, where the average is evaluated on all the possible interpretations of the probabilistic database. This is different from our approach, where all the interpretations of the data which are not consistent with the integrity constraints are discarded. Another difference is that the approach in [15] aims at fixing only the marginal probabilities of the tuples. Thus, it could assign probability 40% to a pair $\langle \tau, l \rangle$, but it would not be able to represent that 39% out of 40% covers the case that the object at $\tau + 1$ is at a location l', while 1% out of 40% covers the case that the object at $\tau + 1$ is at a location l''.

Other related work/research projects are the following:

– [16], where the problem of conditioning probabilistic databases was first addressed. Indeed, our approach is inspired to the formal framework proposed in [16], that we specialized for cleaning RFID-data. In this seminal work, *ws-trees* were introduced as a compact representation of conditioned databases, and algorithms were devised for obtaining a ws-tree starting from a probabilistic database and a set of constraints expressed as ws-sets. However,

directly applying the technique in [16] to our scenario would require to give exponentially large ws-sets (encoding DU, LT and TT constraints) as input to their algorithm for obtaining ws-trees. This makes using the general approach in [16] impractical for conditioning RFID-data.

– the Lahar Project [18, 19, 22]: see Remark in Section 5.

– the Spatial PrObabilistic Temporal (SPOT) framework, which is a general paradigm for reasoning with probabilistic statements about moving objects (see [13] for a survey). In [12], *reachability* rules (corresponding to our DU constraints) have been introduced and exploited in a different problem, i.e., revising a KB representing moving objects when fresh information (in the form of probabilistic spatial-temporal atoms regarding their position) is added to it. The issue of querying SPOT data has been investigated in [21, 20, 11] but without considering data violating reachability rules.

8. CONCLUSIONS AND FUTURE WORK

A probabilistic cleaning framework for RFID data has been introduced, which cleans trajectories by conditioning them to the event that integrity constraints encoding some knowledge about the map and the motility characteristics of the monitored objects hold. Future work will be focused on extending the framework to take into account other forms of correlations, such as those holding in groups of objects moving together, which typically characterize supply-chain scenarios.

9. REFERENCES

[1] Y. Bai, F. Wang, P. Liu, C. Zaniolo, and S. Liu. Rfid data processing with a data stream query language. In *International Conference on Data Engineering (ICDE)*, pages 1184–1193, 2007.

[2] D. Bleco and Y. Kotidis. Rfid data aggregation. In *International Conference on GeoSensor Networks (GSN)*, pages 87–101, 2009.

[3] S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. Sarma. Managing RFID Data. In *International Conference on Very Large Data Bases (VLDB)*, pages 1189–1195, 2004.

[4] H. Chen, W.-S. Ku, H. Wang, and M.-T. Sun. Leveraging spatio-temporal redundancy for rfid data cleansing. In *ACM SIGMOD International Conference on Management of Data*, pages 51–62, 2010.

[5] B. Fazzinga, S. Flesca, F. Furfaro, and E. Masciari. Rfid-data compression for supporting aggregate queries. *ACM Trans. Database Syst. (TODS)*, 38(2):11, 2013.

[6] B. Fazzinga, S. Flesca, E. Masciari, and F. Furfaro. Efficient and effective RFID data warehousing. In *International Database Engineering and Applications Symposium (IDEAS)*, pages 251–258, 2009.

[7] S. Flesca, F. Furfaro, and F. Parisi. Consistency checking and querying in probabilistic databases under integrity constraints. *J. Comput. Syst. Sci. (JCSS)*, 2014.

[8] H. Gonzalez, J. Han, H. Cheng, X. Li, D. Klabjan, and T. Wu. Modeling massive rfid data sets: A gateway-based movement graph approach. *IEEE Trans. Knowl. Data Eng.*, 22(1):90–104, 2010.

[9] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and Analyzing Massive RFID Data Sets. In *International Conference on Data Engineering (ICDE)*, pages 83–88, 2006.

[10] H. Gonzalez, J. Han, and X. Shen. Cost-conscious cleaning of massive rfid data sets. In *International Conference on Data Engineering (ICDE)*, pages 1268–1272, 2007.

[11] J. Grant, C. Molinaro, and F. Parisi. Aggregate count queries in probabilistic spatio-temporal databases. In *International Conference on Scalable Uncertainty Management (SUM)*, pages 255–268, 2013.

[12] J. Grant, F. Parisi, A. Parker, and V. S. Subrahmanian. An agm-style belief revision mechanism for probabilistic spatio-temporal logics. *Artif. Intell.*, 174(1):72–104, 2010.

[13] J. Grant, F. Parisi, and V. S. Subrahmanian. Research in probabilistic spatiotemporal databases: The SPOT framework. In Z. Ma and L. Yan, editors, *Advances in Probabilistic Databases for Uncertain Information Management*, volume 304 of *Studies in Fuzziness and Soft Computing*, pages 1–22. Springer, 2013.

[14] S. R. Jeffery, M. Garofalakis, and M. Franklin. Adaptive cleaning for rfid data streams. In *International Conference on Very Large Data Bases (VLDB)*, pages 163–174, 2006.

[15] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)*, pages 43–50, 2006.

[16] C. Koch and D. Olteanu. Conditioning probabilistic databases. *PVLDB*, 1(1):313–325, 2008.

[17] C. H. Lee and C. W. Chung. Efficient storage scheme and query processing for supply chain management using rfid. In *ACM SIGMOD International Conference on Management of Data*, pages 291–302, 2008.

[18] J. Letchner, C. Ré, M. Balazinska, and M. Philipose. Access methods for markovian streams. In *International Conference on Data Engineering (ICDE)*, pages 246–257, 2009.

[19] J. Letchner, C. Re, M. Balazinska, and M. Philipose. Approximation trade-offs in markovian stream processing: An empirical study. In *International Conference on Data Engineering (ICDE)*, pages 936–939, 2010.

[20] F. Parisi, A. Parker, J. Grant, and V. S. Subrahmanian. Scaling cautious selection in spatial probabilistic temporal databases. In R. Jeansoulin, O. Papini, H. Prade, and S. Schockaert, editors, *Methods for Handling Imperfect Spatial Information*, volume 256 of *Studies in Fuzziness and Soft Computing*, pages 307–340. Springer, 2010.

[21] A. Parker, G. Infantes, J. Grant, and V. S. Subrahmanian. Spot databases: Efficient consistency checking and optimistic selection in probabilistic spatial databases. *IEEE Trans. Knowl. Data Eng.*, 21(1):92–107, 2009.

[22] C. Ré, J. Letchner, M. Balazinska, and D. Suciu. Event queries on correlated probabilistic streams. In *ACM SIGMOD International Conference on Management of Data*, pages 715–728, 2008.

[23] T. Tran, C. Sutton, R. Cocci, Y. Nie, Y. Diao, and P. J. Shenoy. Probabilistic inference over rfid streams in mobile environments. In *International Conference on Data Engineering (ICDE)*, pages 1096–1107, 2009.

[24] F. Wang and P. Liu. Temporal Management of RFID Data. In *International Conference on Very Large Data Bases (VLDB)*, pages 1128–1139, 2006.

[25] J. Xie, J. Yang, Y. Chen, H. Wang, and P. S. Yu. A sampling-based approach to information recovery. In *International Conference on Data Engineering (ICDE)*, pages 476–485, 2008.