Probabilistic Inference of Object Identifications for Event Stream Analytics

Di Wang, Elke Rundensteiner, Richard Ellison III [#], Han Wang
Worcester Polytechnic Institute
{wangdi, rundenst, wanghan}@cs.wpi.edu
[#]University of Massachusetts Medical School
richard.ellison@umassmemorial.org

ABSTRACT

Recent years have witnessed the emergence of real-time object monitoring applications driven by the explosion of small inexpensive sensors. In many real-world applications, not all sensed events carry the identification of the object whose action they report on, so called *"non-ID-ed" events*. Reasons range from heterogeneous sensing devices to human's choosing to conceal their identifications. Such non-ID-ed events prevent us from performing object-based analytics, such as tracking, alerting and pattern matching. We propose a probabilistic inference framework, called FISS, to tackle this problem by inferring the missing object identification associated with an event. Specifically, as a foundation we design a time-varying graphic model to capture correspondences between sensed events and objects. Upon this formal model, we elaborate how to adapt the Forward-backward (FB) inference algorithm to continuously infer probabilistic identifications for non-ID-ed events. However, we demonstrate that FB is neither scalable nor efficient over event streams. To overcome this deficiency, we propose a suite of strategies for optimizing its performance, including the *selective smoothing* technique that significantly reduces the number of random variables that need to be smoothed, and the finish-flag mechanism that enables *early termination* of backward computations. Our experimental results, using large-volume streams of a real-world healthcare application, demonstrate the accuracy, efficiency, and scalability of FISS. Especially FISS achieves on average 15x higher throughput than our basic FB inference.

1. INTRODUCTION

Real-time object monitoring systems are becoming increasingly popular in domains ranging from healthcare, inventory management, public transit management, traffic monitoring to home safety care [3, 24, 25, 35, 37]. These systems receive high-volume event streams from sensors installed at locations of interest. These events then are filtered and correlated for complex pattern detection, aggregated on various temporal and geographic scales, and transformed into high-level actionable information.

In object monitoring systems, while sensed events are often attached with the unique identification of the tagged object (called "ID-ed" events), it is equally common that events may not carry their object identification (called "non-ID-ed" events). Consequently input event streams may be composed of *both ID-ed and non-ID-ed* events. Reasons that cause such mixed event streams include:

- Events may come from heterogeneous sources. While some devices are capable to provide events with object identification, e.g., RF sensors [3, 11, 22], other devices do not capture object identification, e.g., Passive Infrared sensors [14, 28, 33]. Object monitoring applications may consist of heterogeneous monitoring devices, for example both RF sensors and Infrared sensors. For instance in a hospital inventory management system [3], RF taggers are attached to medical equipments to track their distributions, while Infrared sensors are installed in nurse stations and doctor offices to check the sterilization procedure of the equipment. As another example in the hospital environment, it is a regulation to restrict the number of staff in an Operation Room [9] in order to prevent airborne infections to the patient in operation. In some hospitals, Infrared sensors are thus installed in Operation Rooms to check on the number of staff over time, and to observe the door openings to get an insight into airflow transmission. Simultaneously, RF sensors are equipped to also monitor the hygiene performance of healthcare workers [35]. As a result, data collected in those applications will have mixed ID-ed and non-ID-ed events.

- Furthermore, some objects being monitored, usually human beings, may advertently choose to conceal their identification for privacy concerns, or inadvertently forget to show their identification. In some settings, the environmental condition detection, like detecting an object's motion and presence, is only loosely coupled with an object's identification detection. For instance, in real-time hospital infection control systems, such as HyReminder [35], a sensor installed in the hospital has two components: the motion reader that detects a healthcare worker's behaviors, such as sanitizing hands and entering a patient's room; and the badge reader, which records a worker's identification only when she actively presents her badge to the reader. So when a worker shows her badge and then washes her hands, a "wash" event with her identification is generated. Otherwise, if the worker chooses not to present her badge or forgets to show her badge, a "wash" event *without* any identification is generated. Consequently event streams observed in such applications also consist of both ID-ed and non-ID-ed events.

Given such a mixed input stream, those non-ID-ed events pre-

EDBT/ICDT '13 March 18 - 22 2013, Genoa, Italy

vent us from performing object-based analytics, such as object tracing, alerting and pattern matching, which usually is the key service needed in object monitoring applications. For example, the HyReminder system [35] continuously tracks each healthcare worker for hygiene compliance by running a set of pattern queries. An example pattern query is to observe whether a worker cleanses her hands before contacting a patient. These pattern queries are based on events associated with an individual worker. This means we have to know the worker's identification associated with an event before we can correctly utilize the event in the query evaluation process. Therefore, in this paper we address a fundamental data transformation problem for event streams mixed with ID-ed and non-ID-ed events, namely to translate raw streams into queriable, probabilistic event streams with object identification.

State-of-the-art. Recent research on RFID data cleaning and inference [11, 13, 22, 23, 34] assumes that events detected by RF readers are identified by an exact object identification. Instead they focus on issues like cleaning redundant readings and inferring objects' precise locations from several overlapping sensors. In other words, these works tackled fundamentally different problems from our problem of uncertain object identification inference.

On the other hand, the probabilistic data association (PDA) problem is closer to our target problem, as PDA aims to determine the correct correspondence between measurements and objects [6]. The most widely-used approaches to tackle PDA are MHT [31] and JPDAF [33]. These approaches establish probabilistic models based on the fact that in a typical PDA application, such as identifying targets in radar observations or tracking people in a video, the events never carry any object identification. Hence, if we were to apply the PDA techniques to our event stream mixed with ID-ed and non-ID-ed events, they would fail to take advantage of ID-ed events for inference. This then would result in limited precision as confirmed by our experimental analysis (Sec. 6.2). Besides, existing work of PDA largely focused on modeling [6, 31, 33], while the efficiency of processing has been overlooked - which is now a key objective of this paper.

Contributions. To fill the void, we propose a novel probabilistic inference framework called Familiar- Stranger System, or **FISS** [1], that efficiently transforms raw streams of ID-ed and non-ID-ed events into queriable streams of events with probabilistic object identifications. Specifically, our technical contributions include:

Modeling. We devise a time-varying graphic model to capture the underlying event stream generation process from the physical world, including the key component−the temporal correlations among events. In contrast to existing work on either solely ID-ed events [11, 13, 22] or solely non-ID-ed events [31, 33], we now embrace ID-ed and non-ID-ed events within a single model. This keeps our model simple yet while concisely expressing both the true objection identifications (which we may not observe) and the sensed events (which we do observe) (Section 3).

Efficient Inference. Based on our proposed model, we extend a classical inference approach, the Forward-backward algorithm [27], to infer the object identification of non-ID-ed events (Sec. 4). However, our experimental evaluation (Sec. 6.3) demonstrates that this approach, though suitable for our inference logic, is not efficient enough to provide near-real-time system responsiveness nor scalable for a high volume event stream.

Our second contribution is to devise a suite of strategies for optimizing the performance of the Forward-backward inference. Our key insight is that the Forward-backward algorithm conducts a large number of unnecessary computations during the backward smoothing. We aim to avoid such waste by only computing the "affected" events, i.e., events whose distributions should be revised in the backward smoothing. Our first strategy is to *prune random variables* that can be shown to be unaffected by exploiting the features of ID-ed events. The second optimization, called *finish-flags* mechanism, enables early termination of the backward computation yet without sacrificing inference precision. Lastly, we propose to represent temporal conditional dependencies using Complex Event Processing (CEP) *pattern queries* to capture temporal correlations of events in a large volume stream [5, 17, 19, 37]. And then chasing down "affected" events can be transformed into a pattern matching. Meanwhile, we devise an advanced data structure customized for streaming uncertain events to speed up the optimized backward probability computation. These strategies together lead to a solution that keeps up with high-volume streams while offering high-precision inference results (Section 5).

System and Evaluation. Our third contribution is the implementation and thorough performance evaluation of FISS over event streams of healthcare object monitoring. The experimental results demonstrate that our proposed model achieves better inference precision compared to the MHT model [1, 31]. Moreover, our optimization techniques for the Forward-backward algorithm make it work 15 times faster than the basic implementation [2] (Section 6).

2. PROBLEM STATEMENT

Physical world. FISS targets environments with well-bounded sub-spaces, such as an Intensive Care Unit (ICU) with dozens of separated patient rooms, where sensors are installed *within* each room. In this setting, the surveillance areas of sensors *do not overlap*. So an object will be detected by *at most one* sensor at a time. This is a distinct difference from the existing RFID data cleaning literature, where an object can have many redundant readings at a time [11, 22, 23, 34], due to assuming sensors with overlapped surveillance areas.

For ease of exposition, the rest of this paper assumes the environment is an ICU, as depicted in Figure 2a. Such layout is typical in clinics and hospitals. In this paper, we use the HyReminder [35] system, deployed at University of Massachusetts Memorial Hospital, as a representative application. It aims to track, monitor and remind healthcare workers with respect to hygiene compliance. However our techniques are general and can equally be applied to other applications that work with streams mixed with ID-ed and non-ID-ed sensor readings. In summary, the physical world being monitored is an ICU composed by a set of separate rooms $\mathcal{R} = \{R_1, ..., R_{|\mathcal{R}|}\}$ and a set of healthcare workers being monitored, i.e., objects $\mathcal{O} = \{O_1, ..., O_{|\mathcal{O}|}\}$.

To simplify our discussion, we assume the physical world is closed, namely, the number of rooms and number of objects remain constant. We also assume that each object is independent of each other, which is the "disjoint tracks constraint" commonly adopted in the PDA literature [6, 31, 33].

Input event stream. Each sensor in the ICU interrogates objects' movements in its range and immediately returns its reading to the server. The server collects raw readings from all sensors and merges them into a single input stream. We abstract each sensor reading as an *event*. Each event in the stream, denoted by a lowercase e, corresponds to an instantaneous and atomic occurrence of interest [37]. Events are conceptually grouped into *event types*. Every event type is distinguished by its event type name. The event type of e is derived based on the physical information about the sensor. For example, a motion sensor installed over a patient room's door will generate events of type `Exit-patient-room`

[1] A familiar stranger is an individual who we repeatedly observe and yet do not know directly. Our system is given this name because it aims to identify those continuously observed non-ID-ed events.

and Enter-patient-room, whereas a sensor installed at a sanitizer will generate events of type Sanitize. The integrated input stream is mixed, meaning events may or may not have object identifications. Such stream is commonly seen in object monitoring applications for various reasons provided in Section 1.

Each event type has associated attributes as defined by the schema. We assume an event e in the input stream has the schema: event-type (nonce,ts,room#,OID). Here nonce is a unique number attached by the server distinguishable for any events. room# $\in \mathcal{R}$ represents the room of the sensor that generates e, i.e., e's location. As described previously, our target environment is well-bounded and closed, so room# is assumed to be accurate. OID is the object identification associated with e. Since the object identification of e may not be available, we allow OID to be a concrete $O_i \in \mathcal{O}$, or OID $= null$. In the rest of this paper, we denote an ID-ed event as $\hat{e}$, and a non-ID-ed event as $\widetilde{e}$.

Output event stream. The output of FISS is a probabilistic event stream, where each event has an associated probabilistic object identification. As in most probabilistic data management models [4, 8], we represent the probabilistic object identification using the possible-worlds semantics. Namely, the OID of an output event consists of a set of mutually-exclusive *alternatives* with associated *confidence* values. Intuitively, the object identification takes the value of one of its alternatives, and the probability of taking a particular alternative is given by its confidence value. For example, consider an output event enter-patient-room(127,12/01/28 17:30:00,R1,<O_1:0.6,O_2:0.3,O_3:0.1>). This event is most likely associated with object O_1 (60% chance), but may also be with O_2 (30%) or O_3 (10%).

Intuitively, the uncertainty of OID can change over time when additional information is obtained from a newly arrived event. Then the previously inferred object association could be revised to gain a more accurate probability. This revision task is called *smoothing* in probabilistic inference algorithms [32]. Therefore, FISS is designed to also output a special event called *revision*, which represents the modification over a previously inferred OID of an event. A revision has the format: Rev(nonce-of-previous-event, new-OID). Whether to output revisions is an option chosen by the user. In this paper FISS supports three commonly-used output strategies: (i) report any change of identification association obtained during the smoothing; (ii) report when the revision results in 100% confidence; or (iii) report when the revised probability is more than a threshold, say 50% different from the previous probability of that event.

In summary, FISS outputs a stream consisting of probabilistic events and optionally revision events. Such output stream can then be fed into an event management system for object-based analytics. For example, the output stream from FISS for the HyReminder application will be used for hand hygiene pattern detection [35]. Many stream systems are capable to process various queries over such probabilistic streams, including relational queries [23], complex event queries [30] and aggregate queries [18]. Also, several event stream processing systems [5, 7, 26] support revisions that amend previously arrived events.

The inference problem. Finally, the problem we solve in this paper is: given an input stream of raw sensor reading events, where an event may or may not have an associated object identification, we derive a queriable, probabilistic event stream where each event is associated with probabilistic object identification. We aim to infer the identification association as accurately as possible and to do it in an efficient manner so to achieve near real-time responsiveness.

$\hat{e}$	an ID-ed event
$\widetilde{e}$	a non-ID-ed event
s_i	room state variable for Room R_i
ψ_j	object identification association variable for $\widetilde{e_j}$
ϵ_j	information variable for event e_j

Table 1: Summary of Symbols

3. PROPOSED GRAPHICAL MODEL

In this section we present our proposed probabilistic model that captures the correspondences among sensed events and monitored objects. In contrast to existing work on either solely ID-ed events [11, 13, 22] or solely non-ID-ed events [31, 33], our solution embraces ID-ed and non-ID-ed events within a single model. This keeps our model simple yet while concisely describing the physical world.

3.1 Components of the Model

Our model describes the world using random variables that present both true object identification associations, which we may not observe, and the input events, which we directly observe.

Given numerous event types abstracted in the representative HyReminder application, to simplify our discussion, we focus on two important event types Enter-patient-room and Exit-patient-room, or Enter and Exit in short respectively. These two types of events are most critical because they tell us an object's movements throughout the ICU, which serve as the basic knowledge from which we can estimate object identification.

Time and space. Same as most data stream management systems [16, 37], we divide time into a sequence of discrete epochs of, for example, one second in duration. All sensor readings that occur in the same epoch are treated as simultaneous. Each event e can thus be attached with a timestamp from the discrete epoch domain, denoted by e.ts. In our representative application, assuming the epoch granularity is one second, a healthcare worker can conduct at most one action in a single epoch. That is, one object will trigger *at most one* Enter or Exit event at an epoch. As for space, given the well-bounded physical layout we target, it suffices to model the space as a discrete set of rooms, while each room connecting to the hallway, as shown in Figure 2a.

Room state variables. In our problem setting (Section 2), a non-ID-ed event does not carry an objection identification, but does carry its accurate location, in terms of Room#, and timestamp ts. Intuitively, we can utilize the Room# and timestamp as a starting point to estimate the OID of the event. Casually speaking, suppose a non-ID-ed Exit event $\widetilde{e_j}$ occurs at the room R_i, we can figure that those who were in room R_i previously have the possibility to exit the room now. So we will consider those objects as candidates to associate with $\widetilde{e_j}$. Similarly, for a non-ID-ed Enter event, we will look for those objects who were outside the room previously, as they are possible to enter the room at this time. This intuitive observation suggests us to maintain the state of each room, i.e., which objects are in the room at a certain time, so that later we can refer to this information for object identification association.

Therefore, we define a *room state variable* for each room. Namely, for each $R_i \in \mathcal{R}$, let random variable s_i^t present which objects are in R_i at a given time t. Ideally we wish to express a room state variable s_i^t as a set of objects. But because of the uncertainty of OID of an event, a room state could be uncertain as well. In this case each object in the room state will be associated with a confidence value. For example, s_1^{12} = <O_1:0.3, O_2:0.7> represents that for Room R1 at time 12, object O_1 has 30% possibility to be in that room while O_2 has 70% possibility. A room state could also be an empty set when no healthcare worker is spotted in the room. In ad-

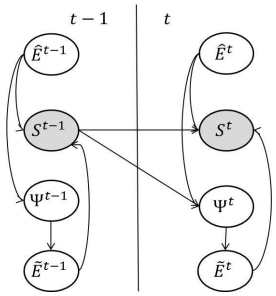


Figure 1: Graphical model for FISS

dition, in our representative ICU layout (shown in Figure 2a), every patient room has a door connecting to the hallway. In our model we treat the hallway as a special kind of room, with a corresponding random variable called *hallway state*, denoted as s_H.

Object association variables. Since our goal is to infer object identification associations for non-ID-ed events, we next define an *object association variable* for each non-ID-ed event $\widetilde{e_j}$, denoted as ψ_j. ψ_j expresses the probability of an object being associated with $\widetilde{e_j}$. It is important to note that the association can be changed over time, because when obtaining additional information from new events, we may revise the association to a more accurate probability. Therefore, ψ_j is a *temporal* random variable, meaning its value evolves over time. Specifically, ψ_j^t represents the object identification association for event $\widetilde{e_j}$ at time t, where t is different from and regardless of $\widetilde{e_j}$'s timestamp $\widetilde{e_j}.ts$. For example, given a non-ID-ed event `enter(122,12,R1,?)`, where 122 is the nonce, 12 is its timestamp and "?" means its `OID` is unknown, then an association variable is first created at time 12, ψ_{122}^{12}. Assume ψ_{122}^{12}= $\{O_1$:0.5,O_2:0.5$\}$, representing that at time 12 we reckon objects O_1 and O_2 have equal chances to be associated with this event. Later at time 14, suppose we are able to improve this association, we set ψ_{122}^{14} = $\{O_1$:0.9,O_2:0.1$\}$, meaning at time 14 we reckon O_1 has 90% probability to associate with this event while O_2 has only 10%.

Event information variables. The input event stream is the source of run-time information that we use to infer object associations. We thus abstract the information conveyed by an input event, e_i, as an *event information variable*, denoted as ϵ_i. Specifically, given an ID-ed event, its information variable presents an object's action (via its event type, `Enter` or `Exit`) and location (via its `Room#` attribute) at time t. In comparison, given a non-ID-ed event, we need to first infer its probable `OID` and then extract its information variable. Consequently, a non-ID-ed event with inferred object association presents one or more objects' probable actions and locations. To discriminate between these two scenarios, let $\hat{\epsilon_k}$ denote an information variable for an ID-ed event $\hat{e_k}$, and $\widetilde{\epsilon_j}$ denote an information variable for a non-ID-ed event $\widetilde{e_j}$.

3.2 Dependencies for Object Association

Next we introduce the conditional dependencies that describe how each component in our model interacts with the other ones, which is used later to compute the probabilities of the object associations. Graphically, the random variables stated previously are represented using *nodes*, and the conditional dependencies presented in this section are represented using *edges*, as depicted in Figure 1.

Room state evolution. This conditional dependency describes how a room state variable, s_i, evolves as new events continuously arrive. Intuitively, at each epoch, the new state is the old state plus objects that are just entering the room, and minus objects that are just exiting the room. In other words, a room state at time t *depends on* its previous state of time t-1 and all objects' movements at time t. Given that objects' movements are provided by newly arrived events, which are modeled as event information variables, it is intuitive that s_i^t conditionally depends on s_i^{t-1} and all event information variables at t. Formally, let $\hat{E}^t$ = $\{\hat{\epsilon_k}|\forall\hat{\epsilon_k}.ts = t\}$ be the vector of all information variables for ID-ed events at t, $\widetilde{E^t}$ = $\{\widetilde{\epsilon_k}|\forall\widetilde{\epsilon_k}.ts = t\}$ be the vector of all information variables for non-ID-ed events at t, then this conditional dependency can be expressed as $p(s_i^t|s_i^{t-1}, \hat{E}^t, \widetilde{E^t})$. Note that the special hallway state s_H is maintained as follows: an `Enter` event at any ordinary room will be considered as an `Exit` event for the hallway, and vice versa.

Object association for non-ID-ed events. Suppose a new non-ID-ed event $\widetilde{e_j}$ occurs at room R_i. Intuitively we know that if $\widetilde{e_j}$ is an `Exit` event, then those who were in room R_i at time t-1, i.e., objects expressed in s_i^{t-1}, will be alternative objects to be associated with $\widetilde{e_j}$. In addition, if there are other ID-ed events simultaneously occurring at t, we can safely exclude those objects from being alternatives of $\widetilde{e_j}$ (because we have assumed that one object can conduct at most one action at an epoch). Formally, this conditional dependency can be expressed as $p(\psi_j^t|s_i^{t-1}, \hat{E}^t)$. Symmetrically, if $\widetilde{e_j}$ is an `Enter` event, then those who were outside R_i at time t-1, i.e., in the hallway, should be the alternate objects associated with $\widetilde{e_j}$. Such objects are listed in the hallway state s_H^{t-1}. So this conditional dependency can be represented as $p(\psi_j^t|s_H^{t-1}, \hat{E}^t)$.

Formal joint model. Now we are ready to state the formal description of our model. To make the notations compact, let S^t = $\{s_i^t|\forall R_i \in \mathcal{R}\}$ be the vector of all the room states at time t, ψ^t = $\{a_j|\forall\widetilde{e_j}.ts = t\}$ be the vector of all object identification association variables at time t. Assume the initial room states S^0 are known. We combine the above conditional dependencies to define a joint model over the entire domain:

$$p(\boldsymbol{\Psi}, \boldsymbol{S}, \widetilde{\boldsymbol{E}}|\hat{\boldsymbol{E}}, \widetilde{\boldsymbol{E}}) = p(S^0)\prod_t\prod_{\widetilde{e_j^t}} p(\psi_j^t|S^{t-1}, \hat{E}^t)$$
$$\prod_{i\in\mathcal{R}} p(s_i^t|s_i^{t-1}, \hat{E}^t, \widetilde{E}^t)$$

where $\boldsymbol{\Psi}$ is the vector of all object identification associations over all times, similarly $\boldsymbol{S}$ is the vector for all room states over all times; $\hat{\boldsymbol{E}}$ is the vector for information of ID-ed events over all times and $\widetilde{\boldsymbol{E}}$ is the vector for the information of non-ID-ed events over all times. Our model can be viewed as a particular case of a Dynamic Bayesian Network (DBN) [27] but with conditional probability functions specially designed for our problem, which is depicted graphically in Figure 1.

4. INFERRING IDS OVER STREAMS: INITIAL EFFORT

As the raw stream is mixed with ID-ed and non-ID-ed events, the task of translating it into a probabilistic stream is treated as an inference process in our work. Inference is essentially to *estimate the true object association* for non-ID-ed events. Formally speaking, our inference task is, from the joint distribution $p(\boldsymbol{\Psi}, \boldsymbol{S}, \widetilde{\boldsymbol{E}}|\hat{\boldsymbol{E}}, \widetilde{\boldsymbol{E}})$, to compute the posterior distribution of all object association variables, given a sequence of information $\hat{E}^{1:T} = \{\hat{E}^1, ..., \hat{E}^T\}$ and $\widetilde{E}^{1:T} = \{\widetilde{E}^1, ..., \widetilde{E}^T\}$. Namely, to compute, for each association variable $\psi_j^t \in \{\Psi^1, ..., \Psi^T\}$, the distribution $p(\psi_j^t|\hat{E}^{1:T}, \widetilde{E}^{1:T})$.

4.1 Inference using FB Algorithm

We first set out to adapt the classical *Forward-backward* (FB) inference algorithm [27, 32]. In this section, we describe the main intuition of how and why the FB inference algorithm tackles our problem. This provides a technical context for our later optimiza-

tions in Section 5.

To find the posterior distributions, i.e., the most likely states for random variables, the FB algorithm involves three steps [27, 32]: (1) computing a set of forward probabilities, (2) computing a set of backward probabilities, and (3) computing smoothed values. Steps (2) and (3) are typically performed simultaneously, called "*backward and smoothing*", which look backward at any past distributions while accounting for the current information, and then obtain more accurate results when possible.

Extension for FB algorithm. The FB algorithm has originally been designed for static data set, we now *extend it to the streaming context*: all arrived events are stored; whenever a new event arrives, the above three steps are performed over all events received so far; the new event is output with (inferred) object identification. Updated values during smoothing may be optionally output as revisions if the user chooses to. In the rest of this paper, we use the term "FB algorithm" to refer to this extended version of FB.

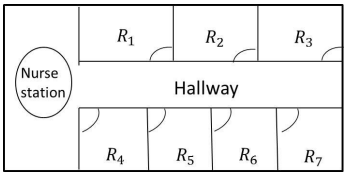
Given that our target applications are real-time systems, typically only the relatively recent events are of interest. Though a revision of a historical event can increase the overall inference correctness, it may not be practically useful at the current time. For example, the HyReminder system would monitor HCW's hygiene behaviors in the last one hour [35]. So a revision for an event more than one hour ago would be assumed not to affect the monitoring results. Therefore, FISS allows user to set an application-specified temporal boundary, call *smoothing window* [22], that restricts the scope of backward revision. The FB algorithm then will only smooth events within this temporal sliding window.

Estimating object associations. Given the conditional dependencies of object association specified in Section 3.2, for a newly arrived event that is non-ID-ed, say $\widetilde{e_j}$, the FB algorithm performs the following in the "forward" step: compute the conditional distribution $p(\psi_j^t|s_i^{t-1}, \hat{E}^t)$ (for an `Exit` event) or $p(\psi_j^t|s_H^{t-1}, \hat{E}^t)$ (for an `Enter` event). Then the resulting distribution is the estimate of `OID` for $\widetilde{e_j}$. Once all non-ID-ed events are associated with probable `OID`s at time t, the FB algorithm keeps room states up to date by computing the conditional distribution $p(s_i^t|s_i^{t-1}, \hat{E}^t, \widetilde{E^t})$.

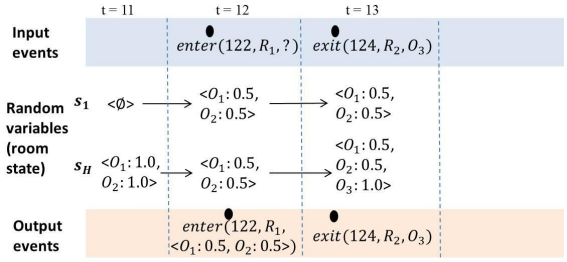
Then in the "backward and smoothing" step, the FB algorithm revisits past distributions while accounting for the fresh information gained at the current time t. Still considering the conditional dependency $p(s_i^t|s_i^{t-1}, \hat{E}^t, \widetilde{E^t})$, but now FB interprets the dependencies in a retrospective manner: if an object, say O_x, exits Room R_i at time t, then O_x would have to have stayed in Room R_i at time t-1. Suppose previously the room state s_i^{t-1} was uncertain about O_x's staying, then the FB algorithm is now able to ensure that O_x must have been in s_i^{t-1} with 100% confidence. As a result, FB amends the previous value of s_i^{t-1}. And then this updated information is carried by FB to compute backward probabilities for other older variables. Since the object association depends on room states, as expressed by $p(\psi_j^t|s_i^{t-1}, \hat{E}^t)$, the revision of a room state may trigger the revision of an object association in turn. Suppose a historic `Exit` event $\widetilde{e}$ at time t depended on the room state s_i^{t-1}, and now s_i^{t-1} is revised. Then FB will revise the `OID` of $\widetilde{e}$ by computing the backward probability of $p(\psi_j^t|s_i^{t-1}, \hat{E}^t)$. The backward and smoothing computation continues going back, one epoch at a time, until reaches the smoothing window boundary.

Next in Example 1, we illustrate how the FB algorithm infers object identifications for non-ID-ed events using the conditional dependencies defined in Section 3.2.

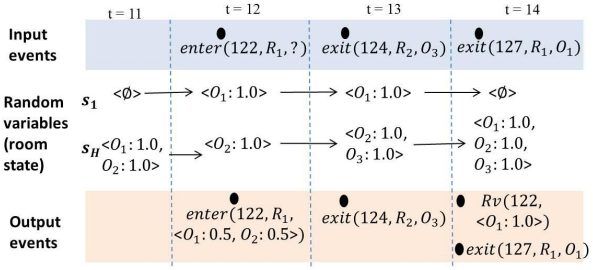
EXAMPLE 1. *We make use of a representative layout of an ICU, as shown in Figure 2a. Figures 2b and 2c depict Room R1's state s_1 and Hallway's state s_H, while Room R2 to R7's states are not shown due to space constraints. We assume there are only three healthcare workers in the ICU, i.e., O_1, O_2 and O_3.*



(a) Example ICU floor plan



(b) Inference results at time 12 and 13



(c) Inference results at time 14

Figure 2: Forward-Backward inference example

Figure 2b depicts Room R1's state s_1 and the hallway's state s_H' starting from time t=11. Namely, at time 11, there was no one in Room R1, and there were two workers, O_1 and O_2, in the hallway.

Suppose at time 12, a non-ID-ed event `enter(122,R1,?)` arrives, where 122 is the nonce, R1 is the room number and "?" means the `OID` is unknown. The FB algorithm first performs the forward inference to assign possible object identifications for this event. Based on conditional dependency $p(\psi_j^t|s_H^{t-1}, \hat{E}^t)$, FB checks the hallway's status at time 11. Intuitively, those in the hallway at t=11 have the possibility to enter Room R1 at t=12. So O_1 and O_2 both are alternatives for this event. To simplify our discussion, we assume O_1 and O_2 have equal chances of entering R1. So FB assigns an equal confidence for them, i.e., 50% vs. 50%. Consequently, a probabilistic event `enter(122,R1,<O1:0.5, O2:0.5>)` is output. And then FB computes room states s_1^{12} and s_H^{12} based on the conditional probability $p(s_i^t|s_i^{t-1}, \hat{E}^t, \widetilde{E}^t)$. Since the `enter` event is uncertain, room states s_1 and s_H at time 12 are uncertain too. Namely, s_1^{12} is set to <O_1:0.5,O_2:0.5>, saying that either worker O_1 or O_2 is at Room R1 at t=12, each with 50% probability. Symmetrically, the hallway state s_H^{12} is set to <O_1:0.5,O_2:0.5>. And then FB goes back in time, namely from t=12 to the beginning of the smoothing window (say t=1), to compute backward probabilities for all random variables (not shown in Fig. 2).

Next at time 13, event `exit(124,R2,O3)` arrives, expressing that worker O_3 exits Room R2 at t=13. This is an ID-ed event, so

no identification association is performed. The FB algorithm then computes Room R3's state and hallway's state at t=13. Next FB performs backward smoothing over all random variables. Note that Room R1's historic states are computed for backward probability, however, none of them is changed given `exit(124,`R_2`,`O_3`)`*. Actually, after the backward probabilistic computation,* s_1^{12} *is left as it was. That is, we are still not sure which one of* O_1 *and* O_2 *was in Room R1 at t=12 and t=13.*

Next at time 14, as shown in Figure 2c, event `exit(127,`R_1`,`O_1`)` *arrives. This ID-ed event conveys that worker* O_1 *exited room* R_1 *at t=14. Intuitively we now figure that* O_1 *must have entered Room R1 previously. By computing the backward probability of* $p(\psi_j^t | s_H^{t-1}, \hat{E}^t)$*, FB is able to amend Room R1's historic states (*s_1^{13} *and* s_1^{12}*) and the hallway's historic states (*s_H^{13} *and* s_H^{12}*) to reflect this finding. Namely,* s_1^{12} *and* s_1^{13} *are both changed to* $< O_1$*:1.0>, meaning it is assured that worker* O_1 *was in room* R_1 *at time 12 and 13. Symmetrically,* s_H^{12} *and* s_H^{13} *are revised to present that worker* O_2 *were in the hallway at time 12 to 13. If the user sets to output revision events, then a revision* `Rev(122,` $< O_1 : 1.0 >$`)` *is produced. This revision event conveys that the historical event with nonce 122 should be associated with* O_1 *with probability 1.0.*

4.2 Discussion of Deficiencies of FB Algorithm

The conventional implementation of the FB algorithm described in Section 4.1 is straightforward and easy accessible (by extending an off-the-shelf AI software). However, it is not able to achieve near-real-time responsiveness, nor to scale to a large volume stream, as we will explain below and also demonstrate by experiments in Section 6.3.

In the conventional implementation of FB, the backward and smoothing step is very computation-intensive. This is because this step computes backward probabilities for every random variable one epoch at a time. Suppose a smoothing window contains *n* epochs, then for *each* random variable, the smoothing step needs to check relevant conditional probabilities *n* times [27].

In fact, we observe that many of these computations are unnecessary. For example, in Figure 2b, the event `exit(124,`R_2`,`O_3`)` occurring at time 13 will not change room state s_1 nor the object association for `enter(122,`R_1`,?)`. But unfortunately the FB algorithm would nonetheless run the backward computation for all random variables. Our intuition is that if the distribution of a historic event's `OID` is not affected by the newly added event, the smoothing will produce an "empty" computational result. If we are able to *skip* probabilistic computations for those "unaffected" events, then the cost of FB can be reduced − henceforth its efficiency can be improved.

5. INFERENCE SPEEDUP: OPTIMIZATION STRATEGIES

To overcome the above deficiencies suffered by the FB algorithm, we now devise three advanced techniques, namely, pruning unaffected variables, early termination of smoothing and selective smoothing, to optimize the backward and smoothing step. These techniques lead to a solution that keeps up with high-volume streams while offering the equally high precision of inference as the classical FB algorithm.

Our intuition is, in order to chase down the random variables that need to be computed during the backward and smoothing, we wish to find out which kind of random variables could ever be affected by the newly updated information.

DEFINITION 1. ***Affected random variable****: a random variable that will be set a different value from its current value by the backward probability computation during the inference.*

5.1 Pruning Unaffected Variables

We first show an important property of random variables in our model, which tells us which random variables are definitely *not* affected. The merit of knowing that a random variable is not affected is that the backward computation can thus safely skip this random variable while guaranteeing to offer the same inference result.

THEOREM 1. *If an alternative of a random variable is associated with 100% confidence, then this alternative's confidence will never be changed by the FB inference in future.*

Proof: In our model, an alternative with 100% confidence of a random variable can be produced by two ways: first, it is directly given as input by an ID-ed event; and second, it is inferred by the FB algorithm. So we prove this theorem case by case. In the first case, the information brought by an ID-ed event is considered as a ground truth which does not depend on any other random variable. Henceforth it will not be changed during the inference process. In the second case, we prove by contradiction. If an alternative's confidence will be changed later, then that means it is still uncertain now. In that case, it would not have had 100% confidence.□

LEMMA 1. *If all alternatives of a random variable are associated with 100% confidence, then this random variable will not be an affected random variable.*

Proof: This lemma is derived from Theorem 1 and Definition 1.□

Lemma 1 is intuitive. Let us consider the random variables in Figure 2b for example. At the beginning, the hallway state at time 11, i.e., s_H^{11}, is assured to be $< O_1$:1.0,O_2:1.0>. This implies s_H^{11} will never be changed by any smoothing afterwards, because we are already 100% sure who were in the hallway at time 11.

Therefore the first optimization we propose is to skip the backward computation for those "unaffected" random variables, i.e., those whose alternatives are associated with 100% confidence. Though simple, this optimization is especially appealing in our target environment where the input stream is mixed with ID-ed and non-ID-ed events. Based on our model, an ID-ed event provides relevant random variables with accurate states. Thus there are potentially a large number of random variables with accurate values in our system. Skipping backward computations for those variables will henceforth significantly reduce processing cost .

5.2 Early Termination Using Finish-Flags

Next, we introduce a second optimization strategy that improves performance by early terminating unnecessary computations in the backward and smoothing process, yet without sacrificing inference precision. We start by presenting the intuition of this mechanism using the following example.

EXAMPLE 2. *Let us consider the random variables in Figure 2b again. Based on Lemma 1, we know that the hallway state variable* s_H^{11} *is unaffected. We further observe that all hallway states earlier than* s_H^{11} *will also never be changed. The intuition is, since the state at time 11 is assured, the smoothing performed at time 11 must have completed all necessary computations for* s_H *with the accurate information already. Thus any smoothing process after time 11 will never change those historical states afterwards. Consequently, it is safe to skip backward computations for all hallway states earlier than time 11.*

Example 2 suggests that for a room state variable, the backward step should be able to skip the computations for all historic states

before an accurate state. The justification for this strategy is given in Lemma 2.

LEMMA 2. *For a room state random variable s_i, if its state at time t, i.e., s_i^t, has all alternatives associated with a 100% confidence, then previous states of s_i that are earlier than t will not be affected by any inference occurring later than t.*

Proof: This feature is due to the Markov property of our model and the construction of the Forward-backward algorithm [27, 32]. When computing backward probabilities for s_i upon the arrival of an event e_j^t, the information carried by e_j^t, i.e., ϵ_j, is passed through a sequence of states of s_i, from the current time t to the beginning of the smoothing window, say t_0. The passing is from one epoch to its contiguously earlier epoch. The backward distribution $p(s_i^{t_k}|\epsilon_j, s_i^{t_k+1}, ..., s_i^t)$ computed at every epoch t_k provides the probability of being in the state of $s_i^{t_k}$ given event e_j^t and all future states. Noting that our model (defined in Section 3) exhibits the Markov property, namely the conditional probability distribution of future states of the process depends only upon the present state, not on the sequence of events that preceded it. So the backward distribution is equal to $p(s_i^{t_k}|\epsilon_j, s_i^{t_k+1})$. If the value of ϵ_j and $s_i^{t_k+1}$ does not change, $p(s_i^{t_k})$ will not change neither. Now that s_i^t is assumed to be unaffected, its contiguously previous state s_i^{t-1} will be unaffected too. Then it can be deduced that all previous states of s_i^t will be unaffected. □

Lemma 2 gives a basis for our **early termination** mechanism, namely to stop the backward and smoothing when a room state with 100% probability is reached. To efficiently implement this mechanism, a mark for room state random variables, called *finish-flag*, is created. A finish-flag is attached to a state when the state is certain. The smoothing process for this random variable stops once a finish-flag is encountered. Suppose there are thousands of historical states before the finish flag, which is realistic in our streaming environment, using the finish-flag then can save thousands of backward probability computations. An example of placing and utilizing finish-flags is described below.

EXAMPLE 3. *Figure 3 shows our optimization techniques applied to the same event stream as in Example 1. Room R1's states and hallway's states are listed at the bottom of Fig. 3. From Example 1 we know that s_1^{11} and s_H^{11} have all alternatives with 100% confidences. Hence two finish-flags are created for s_1^{11} and s_H^{11} respectively at time 11. Therefore the backward and smoothing computation for s_1 or s_H will not check any states earlier than time 11. In comparison, in Example 1, the backward computation goes all the way back until the starting point of the smoothing window, which potentially involves hundreds of historic states.*

Furthermore, when creating finish-flags, we can also dynamically purge the states of random variables by removing those states prior to finish-flags, because those states will never be affected afterwards. **Purging by finish-flags** is a complementary policy to purging by the smoothing window in FISS. Purging by finish-flags is more aggressive and henceforth saves more space. This is important in stream processing where runtime data structures need to be purged to avoid memory depletion [16, 37].

5.3 Selective Smoothing via Pattern Matching

Our proposed pruning (Section 5.1) and early termination (Section 5.2) strategies sift out random variables that are definitely unaffected, which serves as a preliminary round of seeking affected random variables. In this section, we propose a method that declaratively detects affected events.

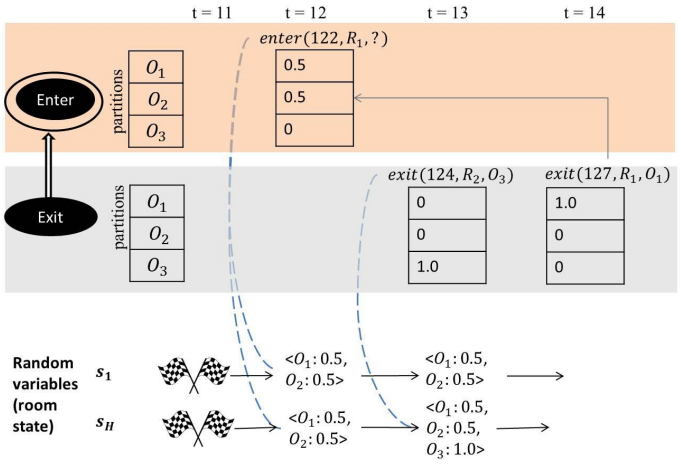


Figure 3: Example of optimized smoothing when *exit(127,R_1,O_1)* just arrives. *Upper*: partitions of probabilistic events. *Bottom*: random variables with finishing-flags. *Dashed lines*: hyper-links from events to random variables.

5.3.1 Motivation of pattern matching based method

We first introduce the notion of affected events, and then present an important observation regarding affected events in our model.

DEFINITION 2. ***Affected event*** *is an event whose object association will be changed by the backward probability computation during the inference.*

LEMMA 3. *An affected random variable depends on at least one affected event.*

Proof: Prove by contradiction. If a random variable depends only on events that are unaffected, then based on the definition of DBN [27], this random variable will not be affected either. This is a contradiction of the assumption that the variable is affected. □

Motivated by Lemma 3, we propose to detect affected events and then follow the conditional dependencies to locate affected random variables, instead of searching through all random variables. There are two main benefits of such event detection based strategy: First, the total number of events in our system is much less than the number of random variables, as can be seen from our model defined in Section 3 and depicted in Figure 1; Second, detecting affected events is more intuitive than detecting random variables, because events are observed by human beings, while random variables are abstractions.

Therefore we now rephrase conditional dependencies $p(\psi_j^t|s_H^{t-1}, \hat{E})$ and $p(s_i^t|s_i^{t-1}, \hat{E}^t, \widetilde{E^t})$ from events' perspective. We found that these dependencies essentially specify the correlations between events, as described below.

OBSERVATION 1. *The backward probability of $p(s_i^t|s_i^{t-1}, \hat{E}^t, \widetilde{E^t})$ tells that if there is an* `Exit` *event for object O_x at room R_i, then there must be a previous* `Enter` *event for O_x at room R_i.* [2]

This observation raises an interesting challenge: can we declaratively specify which event will be affected by a given event? The

[2]Symmetrically, the backward probability also tells that if there is an `Enter` event for object O_i at room R_j, then there must be a previous `Exit` event for O_i at a certain room. However, in our model, entering a room is equal to exiting the hallway, and vice versa. So it is unnecessary to repeat the symmetrical logic in Observation 1.

```
CREATE QUERY Q1
PATTERN SEQ(Enter x, Exit y) ON event-history
WHERE skip_till_next_match(x, y) {
        x.OID ≅ y.OID       AND
        x.Room# = y.Room#       }
WITHIN 1 hour
```

Figure 4: Pattern query Q1 specifying affected event

answer is yes. Intuitively, it appears that an affected event should satisfy the temporal constraint, the event type constraint, as well as the value constraints on OID, as presented in Observation 1. Specifically, our proposed solution is to declaratively specify affected events by using *Complex Event Processing (CEP) pattern queries*.

The reason of choosing CEP pattern queries is two-fold. First, for the sake of *expressiveness*, a CEP pattern query specifies how individual events are filtered and multiple events are correlated via time-based and value-based constraints [17, 19, 21]. This fits our specific problem well. Second, for the sake of *performance*, CEP engines are known for their sophisticated capabilities for detecting temporal correlated patterns of events in huge volume event streams [5, 12, 17, 19, 36, 37]. Thus if we can specify affected events via CEP pattern queries, then the CEP technology can help FISS to detect affected events effectively and efficiently.

5.3.2 Conditional dependency as pattern queries

Next we describe how to declaratively specify affected events using CEP pattern queries based on the conditional dependencies in our model. As a preparation, we describe the properties of an affected event in further depth by extending Observation 1.

LEMMA 4. *Given an event e, suppose event e' is affected by e, then e' should satisfy the following conditions: (1) $e'.ts < e.ts$; (2) $e'.$OID is uncertain; (3) $e'.$OID and $e.$OID contain at least one common alternative; (4) if e's event type is* Exit*, then e''s event type is* Enter*, and vice versa.*

Proof: Conditions (1) and (2) can be directly derived from Theorem 1. We prove condition (3) by contradiction. Suppose the alternative objects for e are $\{O_{j_1},...O_{j_n}\}$. If none of them is an alternative object of e', then that means we are 100% sure e' is not associated with any of $\{O_{j_1},...O_{j_n}\}$. Based on our assumption of "disjoint tracks constraint" presented in Section 2, then we can figure that the association of e does not affect e'. This contradicts the definition of our affected event (Def. 2). For condition (4), this is derived from Observation 1.□

Our **key idea** is to compose a pattern query, using the commonly used CEP pattern query syntax [19, 37], by imposing every conditions in Lemma 4 with proper clauses.

Since the conditions in Lemma 4 require comparisons of probabilistic OIDs, we need to extend the CEP pattern query semantics to support equivalence tests on a probabilistic attribute.

DEFINITION 3. *The **equivalence test over two OIDs**, denoted as ≅, returns true if the two* OID *contain at least one common alternative object, otherwise returns false.*

For example, <A:0.5, B:0.5, C:0> ≅ <A:0.7, B:0, C:0.3> returns true, while <A:0.5, B:0.5, C:0> ≅ <A:0, B:0, C:1.0> returns false.

Next we create the pattern query named Q1 (shown in Figure 4) to express the conditions of affected events as specified in Lemma 4. Q1 exhibits three unique features.

- Q1 utilizes the ***sequence pattern SEQ*** to specify the temporal order in which the events must occur [19, 37]. Namely, when a new Exit event arrives we search for an Enter that occurs previously and satisfies all the predicates in Q1.
- Q1 is defined using the appropriate *event selection strategy* that addresses how to select the relevant events from a large volume of events [21]. Namely, Q1 uses the ***skip-till-next-match selection strategy*** to impose that irrelevant events are skipped until an Enter event matching all constraints is encountered. If multiple events in the event history can match the constraints, only the first one, i.e., the most recent one, is considered.
- In the WHERE clause Q1 specifies predicates on the probabilistic OID attribute, which requires the two events to contain at least one common object alternative (Def. 3).

We have thus shown the successful declarative specification of affected events. This is a transformation from the formal conditional dependencies to practical CEP pattern queries. It is worth noting that *not* all conditional dependencies can be transformed into pattern queries. CEP pattern queries put great emphasis on specifying sequence patterns of events. We thus take this feature to represent temporal conditional dependencies in DBNs, i.e., *temporal arcs* crossing different epochs [27]. For other conditional dependencies that are not temporal, we will use the conventional presentation.

5.3.3 Runtime affected variable detection

Next we describe our proposed algorithm for efficiently detecting affected events and random variables by leveraging CEP pattern matching technology. Specifically, we devise an advanced data structure called *hyper-linked Queue*, or *hyQ* in short, to manage events and random variables in an integrated manner. *hyQ* maintains *partitioned queues* of probabilistic events in order to speed up event pattern matching. It also builds *hyper-links* between events and relevant random variables to help efficiently trace the conditional dependencies.

Initialization. At the initialization stage, suppose all pattern queries created for the model (using the technique presented in Sec. 5.3.2) are registered in FISS. The set of queries is denoted as $\mathcal{Q}$. Following the well-established NFA (Non-deterministic Finite Automata) based event pattern detection mechanism [21, 37], an NFA is created for each pattern query $Q \in \mathcal{Q}$ to represent the sequence of event types. For example, the NFA for Q1 contains event types Enter and Exit, as depicted in Figure 3 by two black ovals. The arrow from Enter to Exit imposes the temporal order between them, namely an Enter event should be ahead of an Exit event, as specified in Q1.

A hyper-linked queue, i.e., *hyQ*, is created for each event type to store events of the same type in time order. In order to expedite the equivalent test on OIDs of events, every *hyQ* is partitioned on the probabilistic OID attribute. The conventional partitioning mechanism in CEP engines [24, 37] dispatches events based on the discrete values of an attribute. However, in our context, OIDs are uncertain. We thus propose to build *partitions based on the alternatives* of an uncertain OID, while recording the *confidence* of each alternative in the partitioned event. This special partitioning mechanism allows an uncertain event to belong to multiple partitions (when its OID has more than one alternative). Figure 3 shows two *hyQ*s for query Q1, namely, the gray block is the *hyQ* for event type Exit, while the pink block is the *hyQ* for event type Enter. Since we assume there are totally three objects in the example application, events in *hyQ*s are partitioned on these three alternatives. Note that event enter(122, R_1, ?) falls into two partitions, O_1

and O_2, because it could be associated with either of these two objects.

For a room state variable, in order to provide efficient ordered access, we index its states by timestamp. Figure 3 illustrates such an arrangement. In the bottom of Figure 3, for example, the state variable for room R1, s_1, has its states organized in time order. Also note that the finish-flags have been established for all historic states where applicable.

Furthermore, to keep track of the probabilistic dependencies, a *hyper-link* is maintained for each event, connecting to all random variables that *depend on* this event. As illustrated in Figure 3, the hyper-link for event `enter(122,`R_1`,?)` points to room R1's state s_1^{12} and the hallway's state s_H^{12}, because the distributions of s_1^{12} and s_H^{12} depend on this event.

Selectively Backward Smoothing. Next we describe the customized backward and smoothing process of FISS. When a new event e arrives, FISS first performs forward inference, same as Step 1 of the classical FB algorithm (Section 4.1). However, FISS then performs the backward computation in a dramatically different way. Specifically, FISS first evaluates all pattern queries in $\mathcal{Q}$ to locate affected events. Suppose the inferred `OID` of e is <O_{x_1} : p_1, ...O_{x_k} : p_k>, where p_k denotes the confidence, and the event type of e is `Exit`. During the query evaluation, FISS only checks those `Enter` events that fall into at least one partition of {O_{x_1}, ..., O_{x_k}}. This evaluation makes heavy use of partitioned hyQs. Namely, for a pattern query Q, its NFA helps to restrict the event type and the temporal order between events. Also its partitioned hyQs enable the equivalence test on probabilistic `OID` to be efficient. The query evaluation returns historic events that are potentially affected by the new event e, while other events will not be considered for further backward computation (because they are guaranteed to be unaffected by our technology). Example 4 illustrates such query evaluation process.

EXAMPLE 4. *In Figure 3, at time t=14, event* `exit(127,`R_1`,`O_1`)` *arrives. FISS follows the NFA to search* `Enter` *events that occurred before time 14. As we can see, all historic* `Enter` *events are stored in hyQs and partitioned by objects. So FISS only needs to check those events falling into the partition of object* O_1*, because* `exit(127,`R_1`,`O_1`)` *is associated with* O_1*. Event* `enter(122,`R_1`,?)` *is quickly detected, which is then determined to be an affected event. And then the evaluation for this particular query stops, because the "skip-till-next-match" clause of Q1 imposes that only the first matched event is returned, as explained in Sec. 5.3.2.*

Next, given an affected event, FISS utilizes the hyper-links to locate affected random variables immediately. Moreover, the pruning (Sec. 5.1) and finish-flag (Sec. 5.2) strategies are *incorporated* during the selection of affected random variables, so that those random variables that are definitely unaffected will be sifted out on the way. Finally, the backward probability computation will account for those affected random variables *only*.

EXAMPLE 5. *In Figure 3, the backward and smoothing at time t=14 only considers Room R1's state* s_1 *and hallway state* s_H*, because they are affected variables of* `exit(127,`R_1`,`O_1`)`*. All other room state variables are not ever accounted for backward computation.*

In summary, our proposed selective smoothing strategy achieves *scalability* by restricting the backward probability computations to a small scope of affected random variables. Most importantly, we do so in a timely fashion by leveraging the CEP technology. This enables FISS to offer the *most likely association* for non-ID-events in *near real-time*, even over a large volume stream.

6. EXPERIMENTAL EVALUATION

We have implemented all proposed inference techniques in FISS using Java and we take the Active CEP framework [36] as our backend CEP engine. In this section, we present a detailed evaluation of FISS using event streams modeled based on the real-world healthcare system HyReminder [35]. All measurements were obtained from a 1.3Ghz Intel Due-core processor with 4GB RAM running JRE 1.6.

Our experimental results demonstrate that our system (1) produces a probabilistic event stream with probabilistic object identifications for all events; (2) offers significant error reduction over the MHT [31] model, a state-of-the-art alternative model; (3) responds to high-volume streaming events within near-real-time on a moderate hardware platform; (4) provides on average 15 times faster processing time than the basic FB algorithm.

6.1 Experimental Setup

Motion tracing event streams. We make use of the data simulator for a hospital ICU scenario that produces sensor reading streams according to the real-world observations obtained in the HyReminder system [35]. Specifically, the simulator has a subroutine that generates a *single* healthcare worker's trace in the ICU, which consists of a sequence of pairs of `Enter` and `Exit` events. Based on our analysis of the real data from the HyReminder system, the time interval of a worker staying at a patient room follows a Gaussian distribution. This subroutine is first executed for each worker separately. Then the simulator *merges* all workers' traces into one stream ordered in time.

Also, to obtain insight into key factors on accuracy and performance, we control several properties of the event stream. One crucial property is the "*non-ID-ed ratio*" of the input stream, meaning the percentile of events that do not have associated object identities. We implement this by randomly selecting a number of events in the stream, for which their `OIDs` are hidden. E.g., if the non-ID-ed ratio is set to 20%, then 20% events in the stream will have their `OID` missing. In this way, FISS cannot see the real `OIDs` of non-ID-ed events, while the simulator keeps a copy of all `OIDs` for later inference precision evaluation.

When we use the simulator to generate healthcare workers' traces, the experimental results below are the average over 20 runs of the stream per each particular setting.

Metrics. Typically, the accuracy of inference results is measured using the *precision* metric, i.e., the ratio of inferred values over the ground truth [6, 22, 34]. In our context, the precision of our inference algorithms can be measured as the ratio of the inferred `OID` over the real `OID` of a non-ID-ed event. For example, suppose the inferred `OID` is $<$ O_1=0.75, O_2=0.25>, and the ground truth is $<$ O_1=1.0>, then the precision is 0.75/1.0 = 75%. Note that we do not calculate precisions for ID-ed events, simply because no object association inference is done for ID-ed events.

The performance metrics are the processing time and the throughput of our system. The throughput is measured as the average number of events that our system takes to process in one unit time. Namely, given a batch of input events of size *numIn* (*numIn* is set to be much larger than the maximum window size of all queries), suppose the system time span taken to process the batch is *Tproc*, then the throughput = $numIn/Tproc$.

6.1.1 Alternative Approaches Compared

MHT. In order to demonstrate our proposed time-varying graphical model provides the desired inference precision, we compare with one of the most widely used approaches for the data association problem, namely the multiple hypothesis tracking (MHT)

model [31]. MHT model maintains multiple possible tracks for each object, and updates every possible existing track when processing a new event. Over time, a track can branch into many possible directions. MHT calculates the probability of each potential track. Typically it only reports the most probable tracks because reporting all is too prohibitive [15]. The main difference between MHT and our proposed model is that MHT does not revise the previously inferred object associations in a track.

In the experiments we adopt an open-source Java implementation [1] of MHT based on the Murty best-k assignment algorithm [15]. The real-world constraints specified in the MHT implementation (in the form of ambiguity matrix and observed features [1]) are equivalent to the conditional probabilities aforementioned in Section 3.2. The MHT implementation is based on the best-k assignment which returns the k most probable tracks for an object at each epoch [15]. We have enumerated k from 100 to 5 and found that a reasonable k in our setting is k=12, which ensures to return results within an affordable time.

Basic FB Algorithm. We compare the performance of our proposed optimization techniques with the off-the-shelf Forward-backward algorithm. We make use of an open-source implementation of FB algorithm in Java [2], and extend the implementation to enable it to work with streaming data, as described in Section 4.1.

6.2 Experiments on Inference Accuracy

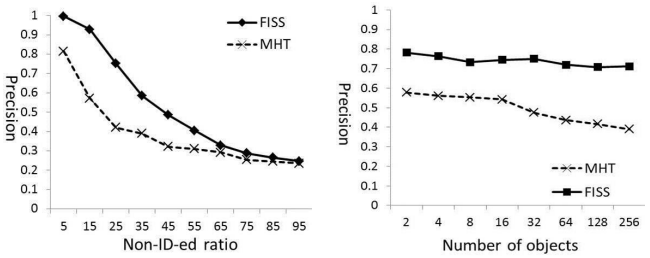
We first evaluate the accuracy of our inference method. Since our optimized inference algorithm and the basic FB algorithm implement the same model, i.e., use the same conditional dependencies specified in Section 3.2, they offer the same inference precision. Thus we only compare our proposed inference technique (marked as "FISS") versus MHT (marked as "MHT") below.

Non-ID-ed ratio vs. precision. We first test the sensitivity to the non-ID-ed ratio of the input stream. As Figure 5a shows, while both FISS and MHT produce worse inference accuracy as the non-ID-ed ratio increases, FISS outperforms MHT by 45% on average. Especially when the non-ID-ed ratio is low, i.e., 5% to 45%, FISS achieves 60% higher precision than MHT. This is mainly because the MHT model does not conduct smoothing over historical object identification associations, even when more information could be gained as new events arrive (especially ID-ed events).

Number of objects vs. precision. Next we vary the number of objects contained in the simulated stream for a fixed non-ID-ed ratio of 25%. Figure 5b reports the average precision results as the object count increases exponentially. We can see that neither of the models degrades significantly. The primary reason is the conditional probabilities specified for object association treat each object independent of each other, which is known as the "disjoint tracks constraint" commonly adopted in the PDA problem [6]. However, MHT scales less gracefully than our FISS approach, especially when the object number is larger than 32. Recall that the MHT implementation is an approximation based on best-k assignment [15], so when the number of objects increases, an non-ID-ed event naturally has more alternate objects that can be assigned to it, yet in MHT only a subset (with the fixed k size) of all possible assignments will be obtained. Such pruning consequently reduces the inference precision of MHT.

6.3 Experiments on Inference Efficiency

Though our optimized inference algorithm and the basic FB algorithm provide the same inference precision, their performances are significantly different. Next we compare the processing time of these two algorithms for consuming the same chunk of the input stream (of 2000 events). The following three experiments (Figures 6 (a), (b) and (c)) convey that our optimized inference algorithm (denoted by "FISS") results in a dramatically less processing time than the basic FB algorithm (denoted by "FB") – while FISS only needs several *thousands* of milliseconds to process, the basic FB approach requires several *millions*, i.e., 15-fold faster.



(a) Vary stream non-ID-ed ratio (b) Vary number of objects

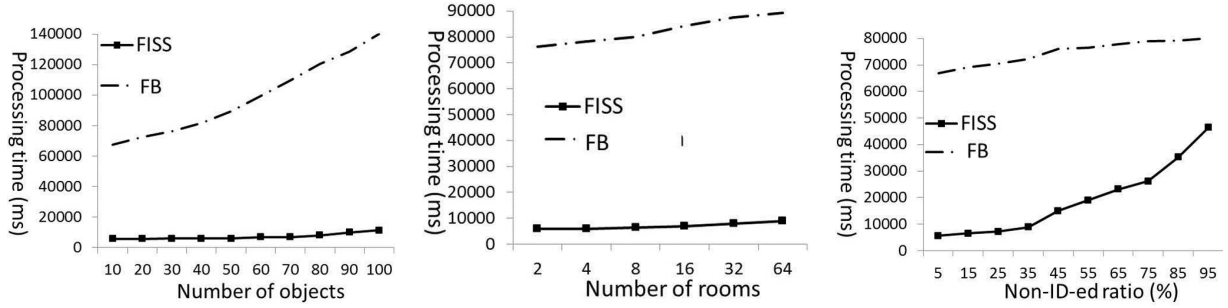
Figure 5: Comparison on Inference Accuracy

Number of objects vs. processing time. In this experiment we vary the number of objects observed in the input stream. The non-ID-ed ratio of the input stream is 25% and number of rooms is 10, with a smoothing window of 60 minutes. As can be seen from Figure 6a, FISS offers a significantly better processing time than the basic FB. Namely, FISS processes the input stream on average 15 times faster. Moreover, we observe that FISS is not very sensitive to the number of objects. The main reason is FISS partitions events on object ID. Even though the total number of objects increases, only the related objects to an event, which typically is fairly stable within a given time period, need to be considered for computation. On the other hand, as explained in Section 4.2, the basic FB algorithm is linear in the number of possible states of a random variable [27], which in our context is exponential in the number of objects.

Number of rooms vs. processing time. Next we observe the processing time of FISS and basic FB while varying the number of rooms in the environment. Figure 6b demonstrates that FISS achieves a stable performance when the room number increases. The key reason is FISS only computes backward probability for *affected* random variables, as stated in Lemma 4. This limits the scope of random variables that must be accounted for computation, which usually relates to one particular room and the hallway. On the other hand, the basic FB algorithm is linear in the room number, as it always revisits every random variables during the backward step and more rooms means more variables.

Non-ID-ed ratio vs. processing time. Figure 6c shows the processing time for increasing non-ID-ratios. FISS needs more processing time when the number of non-ID-ed events in the stream increases for three reasons: First, more non-ID-ed events ask for more forward inference computations; Second, an uncertain event will lead to more affected historical events compared to an ID-ed event; Three, as the number of accurate events decreases, fewer finish-flags can be placed. In contrast, the basic FB algorithm does not perform selective smoothing nor early termination. Its processing time is just slightly affected by the non-ID-ed ratio simply because it performs more forward inference computations to estimate missing `OID`s.

Smoothing window vs. throughput. Next we investigate how FISS and basic FB perform under various smoothing window sizes. In this experiment, the smoothing window varies from 30 minutes to 180 minutes. We set the non-ID-ed ratio to be 25%, the number of objects to be 32, which is realistic for an ICU. Figure 7 demon-



(a) Vary number of objects (b) Vary number of rooms (c) Vary non-ID-ed ratio

Figure 6: Processing Time on Stream of 2000 Events

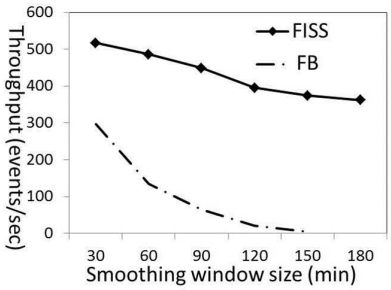


Figure 7: Smoothing window size vs. throughput

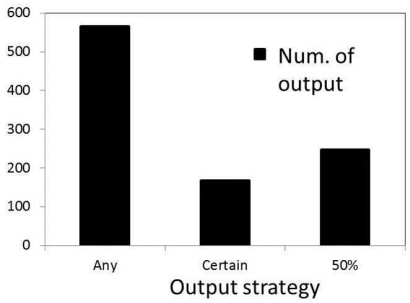


Figure 8: Output strategy vs. num. of output events

strates that both FISS and basic FB degrade in terms of throughput as the smoothing window becomes larger. Because a larger smoothing window means more events must be processed during backward smoothing. But FISS is capable to scale even when the window size is 180 minutes, mainly thanks to the finish-flag technique. As explained in Section 5.2, the optimized backward computation terminates when either a finish-flag is reached or in the worst case, when events are out of the window. This experiment shows that even when the non-ID-ed ratio is relatively large, 25% in this experiment, many backward computations take advantage of finish-flags, thus resulting in much less processing time. In contrast, the FB algorithm conducts each backward revision until the window threshold. That is, its time complexity is polynomial in the number of events [27].

Output strategy vs. number of output events. In this experiment we test three commonly-used output strategies introduced in Section 2, namely reporting any change (denoted as "Any"), reporting when the object association is certain (denoted as "Certain") and reporting when the revision differs by more than 50% (denoted as "50%"). The input stream in this experiment contains 2000 events with 32 objects and the non-ID ratio is 25%. Figure 8 depicts the number of output events, including both ordinary events and revisions, for the three strategies. As we can see, among all changes made during smoothing, shown by the bar of "Any", only a small portion (namely 30%) of them are significant, shown by the bar of "Certain". We also observe that an input event could be modified more than once, resulting in multiple revisions.

7. RELATED WORK

Probabilistic data association (PDA). Techniques for PDA determine the correspondence between measured observations and objects [6] when the association between them is uncertain. In typical PDA applications, like radar based object tracking and person tracking in videos, the input observations *never* carry any object identification. This is the key difference from our target application, where the observations are mixed with ID-ed and non-ID-ed events. Consequently, models and methods from this research area do not work very well for our problem. Our experiments demonstrate that if we adapt the widely-used PDA approach, MHT [31], to our problem, the association results are less accurate than those produced using our proposed FISS solution. Besides, existing work [6, 31, 33] of PDA largely focused on modeling, while the efficiency of processing has been overlooked - which is a key objective of this paper.

RFID stream processing. Recent research has addressed the RFID location inference [11, 13, 34] and RFID data cleaning problems [20, 22]. In these problem settings, object identities are reliably given by RFID readings, i.e., no inference on object identification is needed. Instead, they focus on challenges like cleaning redundant readings and inferring objects' precise locations. Therefore they fundamentally tackle a different problem from us.

Furthermore, in their context, observations of an object are likely redundant, lossy or erroneous. Hence they chose *approximate* inference methods such as particle filtering [11, 13, 20, 22, 34], which maintains weighted samples about the true location of each object. In contrast, in our problem setting (described in Section 2), the information brought by input events are precise, in the sense that even though the identification of an event may be missing, all attributes (including time and location) of an event are accurate. Namely we do not need to handle redundant, lossy or erroneous event streams. We thus chose an *exact* inference method, the Forward-backward algorithm, that fits our problem well. In spite of different models and inference methods, technically we have been inspired by several ideas of efficient inference over streaming data from their

techniques, like limiting the scope of smoothing [22, 34] and customizing data structures to speed up the inference process [11, 34].

Complex event processing (CEP). In recent research on CEP, the focuses have been primarily to enrich the expressiveness of pattern queries and to improve the efficiency of real-time pattern matching [5, 17, 19, 36, 37]. We are *the first* to adopt CEP techniques for probabilistic inference over event streams. We not only provide the methodology of transforming temporal dependencies involved in the inference problem into pattern queries (Section 5.3), but also experimentally show the dramatic performance gain offered by our CEP-aided optimizations (Section 6.3).

8. CONCLUSION & FUTURE WORK

In this paper we present a probabilistic approach to translate a stream consisting of ID-ed and non-ID-ed sensor readings into a probabilistic event stream, thus enabling object-based event analytics in real-time. We design a set of optimization strategies for inferring streaming events using the extended Forward-backward algorithm. The proposed strategies scale the backward probability computation while offering the most likely object association. Our experiments show that our proposed solution offers on average 45% better precision over the state-of-the-art PDA approach. Our optimized inference strategies achieve on average 15 times higher throughput than the basic Forward-backward implementation.

An important finding. Our proposed optimization techniques, namely leveraging pattern queries for selectively smoothing and using finish-flags to early terminate backward computations, are general mechanisms for the Forward-backward algorithm. Even though these techniques were demonstrated in the context of our object identification inference problem, they could be applied to other problems wherever the Forward- backward inference is adopted. Improving the performance of probabilistic inference algorithms with database principles is of independent interest to the probabilistic database community [10, 23, 29]. The generality of our techniques strengthens this paper's contributions and warrants future studies.

In future work, we plan to incorporate our optimization techniques to the real-world HyReminder system. We also plan to explore object-based analytics over probabilistic event streams.

Acknowledgment. This work is supported in part by NSF grant IIS-1018443 and UMMS-WPI CCTS Collaborative grant.

9. REFERENCES

[1] Implementation of the murty algorithm to obtain the best k assignments. http://code.google.com/p/java-k-best/.

[2] Java implementation of algorithms from "artificial intelligence - a modern approach 3rd edition". http://code.google.com/p/aima-java/.

[3] wavemark system: Clinical inventory management solution www.wavemark.net/healthcare.action.

[4] Serge Abiteboul, Paris Kanellakis, and Gosta Grahne. On the representation and querying of sets of possible worlds. SIGMOD, pages 34–48, 1987.

[5] M. H. Ali and C. Gerea et al. Microsoft cep server and online behavioral targeting. *VLDB Demo*, pages 1558–1561, 2008.

[6] Y. Bar-Shalom, editor. *Tracking and Data Association.* Academic Press Professional, Inc., 1987.

[7] Roger S. Barga, Jonathan Goldstein, Mohamed H. Ali, and Mingsheng Hong. Consistent streaming through time: A vision for event stream processing. In *CIDR*, pages 363–374, 2007.

[8] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. *VLDB*, 2006.

[9] J. M. Boyce and D. Pittet. Guideline for hand hygiene in healthcare settings. *MMWR Recomm Rep.*, 51:1–45, 2002.

[10] Hector Corrada Bravo and Raghu Ramakrishnan. Optimizing mpf queries: Decision support and probabilistic inference. *SIGMOD*, 2007.

[11] Zhao Cao, Charles Sutton, Yanlei Diao, and Prashant J. Shenoy. Distributed inference and query processing for rfid tracking and monitoring. *PVLDB*, 4:326–337, 2011.

[12] Sybase CEP. Aleri engine. `http://www.sybase.com/products`.

[13] Haiquan Chen, Wei-Shinn Ku, Haixun Wang, and Min-Te Sun. Leveraging spatio-temporal redundancy for rfid data cleansing. SIGMOD, pages 51–62, 2010.

[14] David Chu, Arsalan Tavakoli, Lucian Popa, and Joseph Hellerstein. Entirely declarative sensor network systems. In *VLDB*, 2006.

[15] Ingemar Cox and Sunita Hingorani. An efficient implementation of reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(2), 1996.

[16] A. Arasu et al. Stream: the stanford stream data manager. *Stanford Tech. Report*, 2004.

[17] A. Demers et al. Cayuga: A general purpose event monitoring system. *CIDR*, pages 412–422, 2007.

[18] G. Cormode et al. Sketching probabilistic data streams. *SIGMOD*, pages 281–292, 2007.

[19] Y. Mei et al. Zstream: A cost-based query processor for adaptively detecting composite events. *SIGMOD*, 2009.

[20] M. J. Franklin and S. R. Jeffery et al. Design considerations for high fan-in systems: The hifi approach. *CIDR*, pages 290–304, 2005.

[21] Y. Diao et al. J. Agrawal. Efficient pattern matching over event streams. *SIGMOD*, pages 147–160, 2008.

[22] S. R. Jeffery and M.J. Franklin et al. An adaptive rfid middleware for supporting metaphysical data independence. *VLDB Journal*, 2007.

[23] B. Kanagal and A. Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. *ICDE*, 2008.

[24] Mo Liu, Elke A. Rundensteiner, and et al. E-cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing. In *SIGMOD*, pages 889–900, 2011.

[25] Eric Lo, Ben Kao, Wai shing Ho, Sau Dan Lee, Chun Kit Chui, and David W. Cheung. Olap on sequence data. *SIGMOD*, pages 649–660, 2008.

[26] Anurag S. Maskey and Mitch Cherniack. Replay-based approaches to revision processing in stream query engines. In *Scalable Stream Processing System*, pages 3–12, 2008.

[27] Kevin P. Murphy. Dynamic bayesian networks: Representation, inference and learning. *Doctor of Philosophy Dissertation*, 2002.

[28] Daniela Nicklas, Matthias Grossmann, and Thomas Schwarz. Nexusscout: An advanced location-based application on a distributed, open mediation platform. In *VLDB*, 2003.

[29] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *VLDB*, pages 373–384, 2011.

[30] Christopher Ré, Julie Letchner, Magdalena Balazinksa, and Dan Suciu. Event queries on correlated probabilistic streams. *SIGMOD*, pages 715–728, 2008.

[31] D. B. Reid. An algorithm for tracking multiple targets. *IEEE Trans. on Automatic Control*, pages 843–854, 1979.

[32] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2009.

[33] M. A. Tinati and T. Yousefi Rezaii. Multi-target tracking in wireless sensor networks using distributed joint probabilistic data association and average consensus filter. In *International Conference on Advanced Computer Control*, pages 51–56, 2009.

[34] Thanh Tran, Charles Sutton, Richard Cocci, Yanming Nie, Yanlei Diao, and Prashant Shenoy. Probabilistic inference over rfid streams in mobile environments. *ICDE*, pages 1096–1107, 2009.

[35] Di Wang and Elke Rundensteiner et al. Active complex event processing: Applications in realtime health care. *VLDB Demo*, pages 1545–1548, 2010.

[36] Di Wang, Elke A. Rundensteiner, and Richard T. Ellison. Active complex event processing over event streams. *PVLDB*, 4(10):634–645, 2011.

[37] Eugene Wu and Yanlei Diao et al. High-performance complex event processing over stream. *SIGMOD*, pages 401–418, 2006.