Real Time Learning of Non-stationary Processes with Dynamic Bayesian Networks

Matthieu Hourbracq[1,2(✉)], Pierre-Henri Wuillemin[1], Christophe Gonzales[1], and Philippe Baumard[2]

[1] Sorbonne Universités, UPMC Univ Paris 6, CNRS, UMR 7606 LIP6, Paris, France
{matthieu.hourbracq,pierre-henri.wuillemin,christophe.gonzales}@lip6.fr
[2] Akheros, Paris, France
{matthieu.hourbracq,philippe.baumard}@akheros.com

Abstract. Dynamic Bayesian Networks (DBNs) provide a principled scheme for modeling and learning conditional dependencies from complex multivariate time-series data and have been used in a wide scope. However, in most cases, the underlying generative Markov model is assumed to be homogeneous, meaning that neither its topology nor its parameters evolve over time. Therefore, learning a DBN to model a non-stationary process under this assumption will amount to poor predictions capabilities. To account for non-stationary processes, we build on a framework to identify, in a streamed manner, transition times between underlying models and a framework to learn them in real time, without assumptions about their evolution. We show the method performances on simulated datasets. The goal of the system is to model and predict incongruities for an Intrusion Dectection System (IDS) in near real-time, so great care is attached to the ability to correctly identify transitions times. Our preliminary results reveal the precision of our algorithm in the choice of transitions and consequently the quality of the discovered networks. We finally suggest future works.

Keywords: DBN · ns-DBN · tv-DBN · Non-stationnary · Learning · Real time · Change point

1 Introduction

In many fields, particularly in information systems and biology modeling, observed processes evolve over time on many scales. Their system states change with time, describing complex trajectories. Some events or entities may influence others at any given time, but those correlations do not necessarily hold forever. Which entity influences another may therefore vary, and any model wishing to capture such a process, without observing the mechanism responsible for such changes, cannot be stationary, that is its structure and/or parameters need to evolve with time too. Otherwise, only one behavior is seen, *averaging* all observations. Since we wish to model the behavior of information systems within a network of computers - in real time - it seems reasonable to assume

J.P. Carvalho et al. (Eds.): IPMU 2016, Part I, CCIS 610, pp. 338–350, 2016.
DOI: 10.1007/978-3-319-40596-4_29

non-stationarity of the observed processes. Indeed, any program or application can accept a wide range of inputs, communicate with other programs, and paths chosen by the process (where it goes and what it does) are often at least input dependent. Thereby, we need a framework for learning non-stationary processes, in real time, and set our focus on non-stationary dynamic Bayesian networks.

Dynamic Bayesian Networks [5,13] are a probabilistic graphical formalism describing, through conditional dependencies, complex dynamical systems under uncertainty. Yet, the use of *dynamic* in DBN refers to the system evolving over time, not the dynamics of the network structure or its parameters. Once determined on a subset of observations, conditional dependencies and parameters are never revisited. In many applications, even more so when data are not produced in a controlled manner, assuming homogeneity of the underlying model(s) describing which state the system is in seems too strong an assumption. This issue has received attention in the last years from the academic field giving rise to non-stationary dynamic Bayesian Networks (ns-DBN) [6–8,17,18] or time-varying dynamic Bayesian Networks (TV-DBN) [20] with applications for system biology [9]. Since processes have many execution paths and a huge input space, it seems unwise to assume that one homogeneous model could accurately capture a process evolution. Two different invocations could result in two completely different traces. Thus we build our system on ns-DBNs.

In this paper, we propose a new algorithm to model non-stationary processes using non-stationary dynamic Bayesian networks. It is organized as follows. We start with (d)BNs and non-stationary dynamic Bayesian Networks. We then build on a non-stationary learning algorithm and present our framework before evaluating its performances on a number of simulated cases to reveal strengths and weaknesses. We finally conclude and extend on our future work.

2 (Non-stationary) Dynamic Bayesian Networks

DBNs are classical Bayesian networks [16] in which nodes $\{X_i(t), i = 1 \dots n\}$, representing (discrete) random variables, are indexed by time t. They provide a factored representation of the joint probability distribution P on a finite time interval $[1, \tau]$ defined as follows:

$$P(\mathbf{X}(1) \dots, \mathbf{X}(\tau)) = \prod_{i=1}^{n} \prod_{t=1}^{\tau} P\left(X_i\left(t\right) \mid \mathbf{U}_i\left(t\right)\right) \tag{1}$$

where $\mathbf{U}_i(.)$ denotes the set of parent nodes of $X_i(.)$ and $P\left(X_i\left(t\right) \mid \mathbf{U}_i\left(t\right)\right)$ denotes the conditional probability function associated with random variable $X_i(t)$ given $\mathbf{U}_i(t)$. $\mathbf{X}(t) = \{X_1(t), \dots, X_n(t)\}$, is called a "slice" and represents the set of all variables indexed by the same time t. This joint probability $P(\mathbf{X}(1), \dots, \mathbf{X}(\tau))$ represents the beliefs about possible trajectories of the dynamic process $\mathbf{X}(t)$.

DBNs assume the *first-order Markov property* which means that the parents of a variable in time slice t must occur in either slice $t-1$ or t:

$$\mathbf{U}_i(t) \subseteq \mathbf{X}(t-1) \cup \mathbf{X}(t) \backslash X_i(t) \tag{2}$$

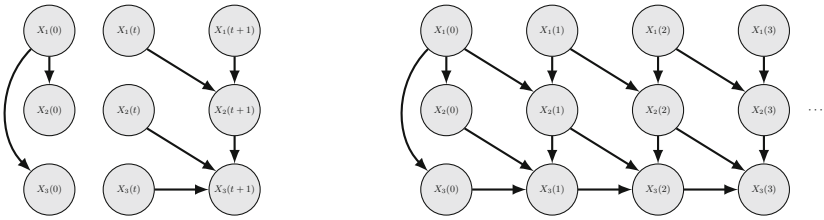

Fig. 1. A 2-Time-Slice BN (2TBN) and the (unrolled) dynamic Bayesian network.

Moreover, the conditional probabilities are time-invariant (*first-order homogeneous Markov property*):

$$P\left(X_i\left(t\right) \mid \mathbf{U}_i\left(t\right)\right) = P(X_i(2)|\mathbf{U}_i(2))), \forall t \in [2, \tau] \tag{3}$$

Hence, to specify a DBN, we only need to define the intra-slice topology (within a time slice), the inter-slice topology (between two time slices), as well as the parameters (*i.e* conditional probabilities, see Eq. 3) for the first two time slices. We obtain a 2TBN such as in Fig. 1.

In this paper, we consider that $X_i(t)$ are all discrete variables and let P^t_{ijk} be the probability that $X_i(t) = k$, given that its parents have instantiation j, *i.e.*

$$P^t_{ijk} = P\left(X_i(t) = k \mid \mathbf{U}_i(t) = j\right), \begin{array}{r} i = 1, \ldots, n \\ j = 1, \ldots, c_i \\ k = 1, \ldots, r_i \end{array} \tag{4}$$

where r_i is the number of values that node $X_i(t)$ can take and c_i is the number of distinct configurations of $\mathbf{U}_i(t)$.

DBNs have been applied in a variety of domains such as speech recognition [12], fault detection [11], medical diagnosis [4] or system biology [19] but their applications on Intrusion Detection Systems are rare [1]. However, (Hidden) Markov Models have been extensively proposed to model system call traces and shell commands [23,24] or network data flow [15]. Bayesian Networks are mainly used in this field in a static manner and often for classification purposes, as a deciding mechanism aggregating smaller models outputs that offer a summary of input data [10,14]. A variant of dynamic Bayesian Networks, called Continuous Time Bayesian Networks (CTBNs), has been used to model network traffic [21]. CTBNs leverage continuous time to solve the issue of time-granularity when using DBNs, which require a time-slice width, thus making them computationally inefficient when dealing with long period of "inactivity" or irregularly spaced observations. The main drawbacks of the framework are that two variables cannot change states simultaneously and a parameter needs to be chosen to scale timing correlations. The work in [21,22] is close to ours in approach; the use of a hidden variable allows to model the machine unknown state - the structure is manually specified and does not evolve. After training, they use a sliding window and selection by likelihood threshold to flag anomalous behavior. However,

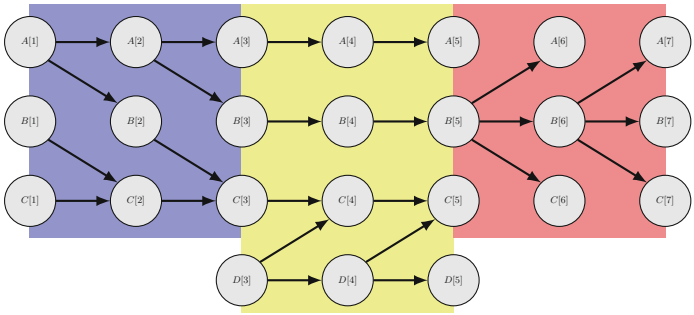

Fig. 2. Non-Stationary dynamic Bayesian network (ns-DBN) with 3 different epochs. Note that DBNs in different epochs may have different parameters, structures and even variables. For simplicity's sake, priors BNs as in Fig. 1 are not represented.

the number of states of the hidden variable needs to be known in advance (two in this case) and new models cannot be discovered on the fly. There is also no mechanism for windows overlapping events from different models.

ns-DBNs are dynamic bayesians networks $\mathcal{B} : (\mathbf{\Theta}, \mathcal{G})$ organized by epochs of varying size or transition times $\mathcal{T} : \{(\mathcal{B}_m : (\mathbf{\Theta}, \mathcal{G})_m, \mathcal{T}_m)\}$. There is no framework yet to model the behavior of the transition times for ns-DBNs. As a consequence, ns-DBNs represent non-stationary processes assuming piece-wise stationarity over epochs, which seems a reasonable assumption. They inherit all advantages and inconvenients of DBNs.

Although different epochs in ns-DBN may alter parameters, structures and even set of variables for the DBNs (see Fig. 2), [7] focuses on parameters evolution with fixed structure. In [18], the focus is set on structural evolution. For this later papers, the number of variables and their domains remain constant over time (even if some are not observed during whole epochs). [6] is close to our approach allowing structure, parameters as well as variables and their domains to evolve over time. However we use different criteria and a mechanism for windows overlapping events from different models to refine transition times.

ns-DBNs learning algorithms consist in identifying the different epochs and the DBNs associated with each. Current learning algorithms focus on either structure or parameter evolution (mainly to cope with the size of the search space). Sadly they pretty much all require the availability of the whole database and cannot be used in our online framework except. Offline learning of ns-DBNs is usually achieved through the use of an updated traditional DBNs scoring function used to account for the sufficient statistics that need to be specified by epoch to find the best time transitions and an updated structural move set for learning the structure which also need to be specified by epoch. However, we need simpler schemes to achieve real time performance while streaming observations. [6] proposes an interesting avenue: to take into account arcs strength using the mutual information of a node and its parent and to use previous parameters as priors for nodes that do not change from one model to another. Further inquiry

is required since information can flow differently in the network according to which nodes are observed.

3 Learning Non-stationary Processes

We present in this section a new algorithm to learn non-stationary dynamic Bayesian networks in real time. In previous literature [7,8,18], the assumption that two adjacent models are governed by similar distributions and/or similar structures is often made. However we do not restrict ourselves to smooth evolutions from model to model.

Real-time data are streamed in a continuous manner using a sliding window. For each new window of data, our algorithm has to choose between using an already known model or creating a new one. This choice is based on the likelihood of the windowed data. Indeed it is expected that the likelihood of a model will decrease if the underlying behavior changes (see Fig. 3). The algorithm begins with a burn-in resulting in a first network that serves as a starting point.

More formally, at any current time τ, the algorithm confronts a collection of M DBN models $\{\mathcal{B}_m : (\mathbf{\Theta}, \mathcal{G})_m\}_M$ with the windowed data $\mathbf{w}[\tau, \tau + r]$.

3.1 Learning with Fixed Variables and Static Window Size

To evaluate how a model m is able to explain the data $\mathbf{w}[\tau, \tau+r]$, we use a simple criterion based on the likelihood on $\mathbf{w}$. In a stationary DBN, the log-likelihood of the data against a network with structure $\mathcal{G}$ and parameters $\mathbf{\Theta}$ is:

$$LL(\mathbf{w} : \mathbf{\Theta}, \mathcal{G}) \propto \sum_{t=\tau}^{\tau+r} \sum_{i,j,k} N_{ijk} \log(\theta_{ijk}) \tag{5}$$

with $\theta_{ijk} = P\left(X_i(t) = k \mid \mathbf{U}_i(t) = j\right)$ and N_{ijk} the number of cases where $X_i(t) = k$ and $\mathbf{U}_i(t) = j$ in $\mathbf{w}$.

For each DBN $(\mathbf{\Theta}, \mathcal{G})_m$, we then compute $LL(\mathbf{w} : \mathbf{\Theta}_m, \mathcal{G}_m)$. However the best matching model cannot be selected only by maximizing LL since the algorithm may also discover new models on the fly. For this purpose, one can note that the distribution of the log-likelihood of a window $\mathbf{w}$ is approximately normally distributed (as a sum of $r + 1$ i.i.d random variables using the central limit theorem). We then design a statistical hypothesis test in order to find the log-likelihood p-value LL_{tr} such that 99 % of matches occur with greater or equal log-likelihood (see Fig. 3). To produce a first estimate of LL_{tr} after discovering a new network, we use Gibbs sampling [3]: trajectories are sampled from the network, as many observations as needed to fit several windows, before computing their likelihood by sliding the window. For each model m, we compute $\frac{LL_{tr}(\mathbf{\Theta}_m, \mathcal{G}_m)}{LL(\mathbf{w}:\mathbf{\Theta}, \mathcal{G})}$. Our selection rule becomes:

$$\arg\max_m \left\{ \frac{LL_{tr}(\mathbf{\Theta}_m, \mathcal{G}_m)}{LL(\mathbf{w} : \mathbf{\Theta}_m, \mathcal{G}_m)} \geq 0.97 \right\} \tag{6}$$

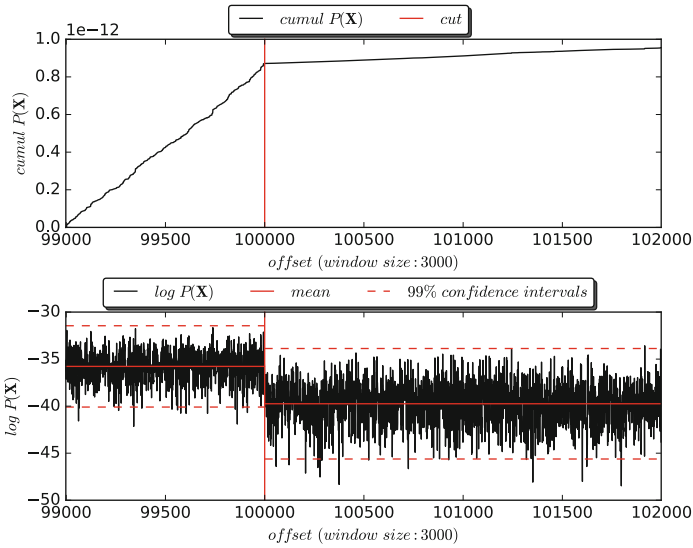

Fig. 3. Cumulative $P(\mathbf{X})$ and $log\ P(\mathbf{X})$ for a window covering two different behaviors. The vertical red line is a transition between models, and horizontal red lines are Gaussian with 99 % confidence interval. (Color figure online)

We use 0.97 as the threshold on the likelihood ratio instead of 1 since the first learning are quite inaccurate - a few events to learn a lot of parameters - and we allow some divergence to occur. A time varying threshold on the likelihood ratio could be designed to take into account parameters and structure convergence. The higher the threshold the more specific the discovered and learned networks will be (as a side-effect we will have more networks, for a given database, than with a lower threshold).

If the set of models in Eq. 6 is empty, the algorithm will learn a new DBN from the window and select it for the current window. If an existing model m is selected, there is still a learning phase in order to update the parameters and eventually the structure with the new data. Indeed, as observations increase for the models, their structures will need to be reevaluated: at each order of magnitude, we then re-estimate the network structures.

In experiments, Fig. 6 shows how a badly sized window can mislead the learning. We therefore propose to adapt the window.

3.2 Dynamic Adaptation of the Window

When the algorithm predicts for the current window a model m_t different from model m_{t-1} of the last window (i.e. m_t would be a newly created DBN or an already existing DBN), the prediction is not only about the change of behaviors of the dynamic process but also about the time τ of this change (the change

point). In this section, we propose to investigate more exactly the value of this point by looking at the distribution of the likelihood within $\mathbf{w}[\tau - r, \tau + r]$.

Figure 3 shows the cumulative value of $P(\mathbf{X})$ and $log\ P(\mathbf{X})$ for such a window (with change point $c = 100\,000$). In order to identify a correct value for c, one could rely on the change of slope in the cumulative $P(\mathbf{X})$. To be more accurate, we maximize over c the likelihood of a model where c separates two different Gaussian processes. We then update m_{t-1} on $\mathbf{w}[\tau - r, \tau - r + c^*]$ and update or learn m_t on $\mathbf{w}[\tau - r + c^*, \tau + r]$ with the optimized change point c^*. If m_t is a new model, we use a non-informative Dirichlet prior, making the assumption that parameters and structure evolve without correlations from one model to another.

3.3 Learning with Incompatible Variables Domain

As seen in Fig. 2, the number of variables may change during the process. In this case, one may have to confront a model and a database with a different numbers of variables. If the variables of the database form a sub-set of the variables of the model, with variable X_e in $\mathcal{G}_m$ but not in the database, we use inference to estimate $P\left(X_i \mid \mathbf{U}_i \setminus X_e\right)$ and then compute the likelihoods. On the other hand, if the variables of the model form a sub-set of the variables of the database, those informations in the database are not exploitable for this model and then are simply discarded. Such a model will not be selected for the current window. If variables domains Ω_{X_i} differs, we add the missing states

Algorithm 1. Main loop

Data: previous model id m^*_{t-1}, observations $\mathbf{w}[\tau - r, \tau]$, $\mathbf{w}[\tau, \tau + r]$
Data: $\mathbf{D} = \{\mathcal{B}_m\}$, $\mathcal{B}_{m^*_{t-1}}$

begin
  $\Phi \leftarrow find\_match(\mathbf{D}, \mathbf{w})$
  if $\Phi \neq \{\}$ **then**
    $m^*_t \leftarrow \arg\max_m \left\{ \frac{LL_{tr}(\mathbf{\Theta}, \mathcal{G})}{LL(\mathbf{w}:\mathbf{\Theta}, \mathcal{G})} : (LL, LL_{tr}, m) \in \Phi \right\}$
    if $m^*_t \neq m^*_{t-1}$ **then**
      find the change point c on $\mathbf{w}[\tau - r, \tau + r]$
      update and validate previous model $\mathcal{B}_{m^*_{t-1}}$ on $\mathbf{w}[\tau - r, \tau - r + c]$
    if *new observations* $\geq$ *10 * previous observations* **then**
      update $\mathcal{B}_{m^*_t}$ structure and parameters with $\mathbf{w}[\tau - r + c, \tau + r]$
      previous observations $\leftarrow$ *previous observations* + *new observations*
    else
      update $\mathcal{B}_{m^*_t}$ parameters with $\mathbf{w}[\tau - r + c, \tau + r]$
    return
  find the change point c on $\mathbf{w}[\tau - r, \tau + r]$
  update and validate previous model $\mathcal{B}_{m^*_{t-1}}$ on $\mathbf{w}[\tau - r, \tau - r + c]$
  learn new model on $\mathbf{w}[\tau - r + c, \tau + r]$

Algorithm 2. Find match

Data: observations $\mathbf{w}[\tau, \tau + r]$
Data: $\mathbf{D} = \{\mathcal{B}_m\}$, $Dir(\{\alpha_{ijk}\})$-(*Dirichlet parameters*)

begin
  $\Phi \leftarrow \{\}$
  for $\mathcal{B}_m = (\mathbf{\Theta}, \mathcal{G})_m \in \mathbf{D}$ **do**
    while $\exists X_e \in \mathcal{B}_m, X_e \notin \mathbf{w}$ **do**
      $\forall j \in [\![1, c_e]\!]$, eliminate X_e using inference :
      $P(X_i \mid (\mathbf{U}_i \setminus X_e) = j)$
    while $\exists X_e \in \mathbf{w}, X_e \notin \mathcal{B}_m$, **do**
      discard X_e
    while $\exists X_i \in \mathbf{w}, X_i = k$ **and** $X_i \in \mathcal{B}_m, k \notin \Omega_{X_i}$ **do**
      $\Omega_{X_i} \leftarrow \Omega_{X_i} \cup k$
      $\theta_{ijk} \leftarrow \frac{\alpha_{ijk}}{N_{ij}+\alpha_{ij}}$
      $\theta_{ij\{o \neq k\}} \leftarrow \frac{N_{ijo}+\alpha_{ijo}}{N_{ij}+\alpha_{ij}}$
      for $X_l \in \mathcal{B}_m : X_i \in \mathbf{U}_l$ **do**
        $\forall j \in [\![1, c_i]\!]$, compute using inference :
        $P(X_l \mid (\mathbf{U}_l \setminus X_i) = j, X_i = k) \leftarrow P(X_l \mid (\mathbf{U}_l \setminus X_i) = j)$
    if $\frac{LL_{tr}(\mathbf{\Theta}, \mathcal{G})}{LL(\mathbf{w}:\mathbf{\Theta}, \mathcal{G})} \geq 0.97$ **then**
      $\Phi \leftarrow \Phi \cup (LL, LL_{tr}, m)$
  return Φ

using the (non-informative) Dirichlet priors α_{ijk} parameters and then compute the likelihoods.

Algorithms 1 and 2 describe our framework for online learning of non-stationary processes with ns-DBN. While the next section will investigate our experiments, it is noteworthy that the complexity of our algorithm does not depend of the size of the database but only of the size of the window and the number of known models which is an important quality for online learning.

4 Experiments and Results

Our experiment consists in modeling simulated non-stationary processes. Using the aGrUM library (http://agrum.lip6.fr), we generated a DBN of 10 nodes by time-step of average domain size 7 ($[\![3, 10]\!]$) and average node degree 3. We then perturbed the structure and parameters of the model using the hellinger distance [2] between the two models as stopping criterion. Multiple thresholds were used to see how far apart two networks need to be for them to be recognized as two independent models. Hellinger distances greater than 0.8 always involve changes in parameters for all nodes and sometimes structure for a few set of nodes. Hellinger distances under this threshold involve parameter changes for one or two nodes, with small degrees, and sometimes an arc is added, adding very little information.

The databases were then sampled from each model before being combined to form a unique dataset consisting of 600.000 events. Different epoch sizes were used in order to see the impact of sample size against network distance as well as different resolutions of the sliding window to see how the system performs when overlapping datasets from two distinct models (i.e. the epoch is not a multiple of the window size). We ran each settings with and without the dynamic window scheme. It is important to note that our algorithm have no prior information about the number of networks, their variables and variables domains or the number of transitions.

The fictive Fig. 4 explains how to read experiments' figures and tables, where *FN* stands for transitions false negatives (percentage of missed transitions over all true transitions), *FP* stands for transitions false positives (percentage of false transitions over all discovered transitions) and *TP* stands for transitions true positives (percentage of true transitions over all discovered transitions). Also, tp is the number of (true) events learned by correct networks, fp the number of (false) events learned by incorrect networks and fn the number of (true) missed events by networks that are learned by others. Adaptive windows can be seen with curves being extended either on the left (for the current matching model moving the window) or the right (non matching models that do not move the window). In experiments' tables, cuts average, minimal and maximal errors are shown, with standard deviation. Finally, *precision* $tp/(tp + fp)$ and *recall* $tp/(tp + fn)$ for events are also shown, that is average *precision* and *recall* over discovered networks. *Recall* amounts to the percentage of correct events found for all correct events that should have been found. *Precision* is inversely proportional to noise (events generated from another model used to update the current model). Due to pages restriction, results were averaged for all thresholds of hellinger distance. We focus on the cases with the epoch not being a multiple of the window size - and show a best case (Fig. 5) and worst case (Figs. 6, 7, 8 and 9) scenario with and without the adaptive window. The results for static and adaptive windows are presented in Tables 1 and 2, respectively.

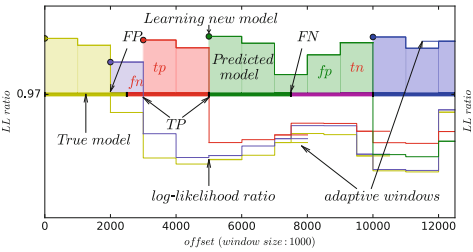

Fig. 4. How to read figures.

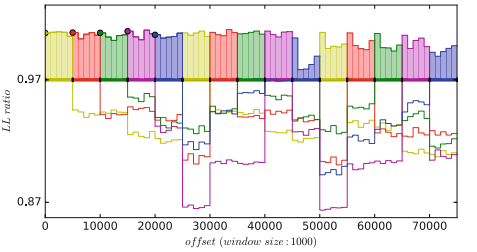

Fig. 5. Results for epochs of $5K$ observations, $hellinger < 0.8$, fixed window size

4.1 Static Windows

Figure 5 is an example of a successful run: the epoch is a multiple of the window size, consequently the sliding window always contains observations from one model at a time. In such settings, correct transition times and models are always identified, with and without adaptive window. However, errors arise when using arbitrary window sizes without dynamical windows as shown in Figs. 6 and 7.

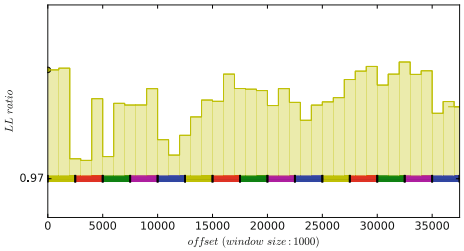

Fig. 6. Results for epochs of $2K5$ observations, *hellinger* < 0.8, fixed window size

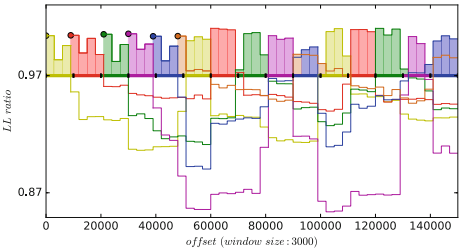

Fig. 7. Results for epochs of $10K$ observations, *hellinger* < 0.8, fixed window size

In the static case as in Table 1, two issues explain the poor *precision* and *recall* for some experiments. The first issue arises when we have discovered fewer networks than we should, mainly with lower hellinger thresholds, in which case transitions were missed and some models are averaging several true models, increasing noise and making further transitions harder to detect, hence increasing *FN* of transitions and decreasing *precision* and *recall* over events. Such a case is highlighted by Fig. 6 and by the first two rows of Table 1, with the first row and Fig. 6 showing results for close true networks and the second row results for distinct true networks. The second issue arises when we have discovered more networks than we should, mainly with higher hellinger thresholds. When it happens, most networks in excess were made when the window overlaps events from two true networks, thus modeling the transition itself (the next window matches or creates another model, the true one), such as in Fig. 7 (the brown network). Hence, we have two transitions instead of one, increasing *FP* for transitions. *Precision* and *recall* are less affected by those *FP* since only a few transitions give rise to very specific models, slightly reducing the *recall* of other discovered (true) networks, but increasing their *precision* (reducing noise).

Results in the static case could be worse: in our setting, one epoch is not a period of the window size but a multiple of the epoch can be a multiple of the window size, in which case the window ends or starts at a true transition, therefore "increasing" our probability of correctly identifying a transition or model. Thus, *FN* and cuts errors could be higher whereas *precision* and *recall* could be lower.

Table 1. Results for static windows, showing missed transitions over all true transitions (false negatives *FN*), false positive transitions (*FP*) and true positive transitions (*TP*) over all discovered transitions. For cuts, minimal, average and maximal error in events, with variance. For discovered networks, *precision* and *recall* over events.

epoch	*window size*	*FN*	*FP*	*TP*	*avg. error*	*std. deviation*	*min*	*max*	*precision*	*recall*
2500	1000	1.0	0.0	0.0	*NA*	*NA*	*NA*	*NA*	0.2	1.0
2500	1000	0.0	0.0	1.0	251.046	249.998	0.0	500.0	0.829	0.875
2500	2000	0.602	0.0	1.0	521.052	361.644	0.0	1000.0	0.5	0.952
5000	1500	0.101	0.035	0.965	351.216	258.546	0.0	1000.0	0.833	0.935
5000	3000	0.0	0.06	0.94	752.1	579.236	0.0	2000.0	0.856	0.854
10000	1500	0.0	0.131	0.869	381.356	295.694	0.0	1000.0	0.961	0.96
10000	3000	0.1017	0.0083	0.992	772.81	601.843	0.0	2000.0	0.833	0.929
15000	2000	0.0	0.204	0.795	512.82	499.835	0.0	1000.0	0.963	0.958

4.2 Adaptive Windows

The results for adaptive windows, shown in Table 2, reveal that the size of the window has little impact on the correct identification of transitions and models, and it should hold as long as the window size is lower than the epoch. Surprisingly, results are not worse for small epochs given the domain size of the network. With adaptive windows, both previous issues are solved by looking for a change point, as in Figs. 8 and 9: in the first case, we do not learn from overlapping windows which reduces noise, making future transitions easier to discover. In the second case, looking for a change point itself avoids the creation of a network to represent the transition alone.

The ability of the algorithm to add modalities to known variables avoids the creation of unnecessary networks in both settings, thus reducing *FP* for transitions, and is of crucial importance for outliers that happen every now and then.

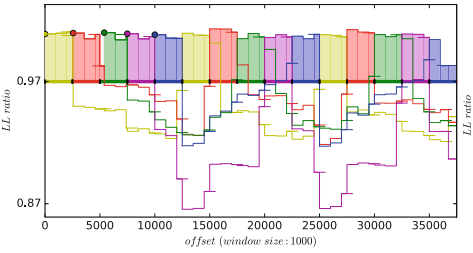

Fig. 8. Results for epochs of $2K5$ observations, $hellinger < 0.8$, dynamic window size

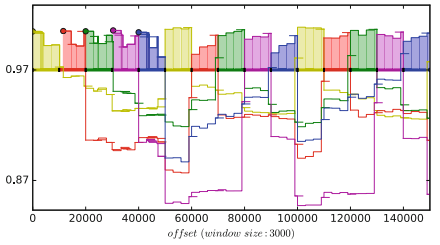

Fig. 9. Results for epochs of $10K$ observations, $hellinger < 0.8$, dynamic window size

Table 2. Results for adaptive windows, with columns as in Table 1.

epoch	*window size*	*FN*	*FP*	*TP*	*average error*	*std. deviation*	*min*	*max*	*precision*	*recall*
2500	1000	0.0	0.0	1.0	4.399	18.045	0.0	254.0	0.998	0.998
2500	2000	0.0	0.0	1.0	2.435	4.683	0.0	38.5	0.999	0.999
5000	1500	0.0	0.0	1.0	3.0966	7.784	0.0	62.5	0.999	0.999
5000	3000	0.0	0.0	1.0	8.702	43.383	0.0	390.5	0.998	0.998
10000	1500	0.0	0.0	1.0	32.923	80.526	0.0	311.5	0.997	0.996
10000	3000	0.0	0.0	1.0	19.559	103.199	0.0	778.0	0.998	0.998
15000	2000	0.0	0.0	1.0	8.551	32.423	0.0	202.5	0.999	0.999

5 Conclusions and Future Work

We built a framework around Dynamic Bayesian Networks to learn non-stationary processes in a continuous manner, designed to be fast and accurate. However, several enhancements comes to mind: we mentioned a dynamical threshold on the log-likelihood ratio to take into account convergence, as well as the need for merging and deleting schemes, since we expect results to be poorer the closer the original networks are from each other. While a naive deleting scheme could consists of using a parameter for each network, decreasing over time when not matching, merging networks require to compare their joint probability distributions which involves heavy computations. The problem of looking for a cut could also be investigated further, since if the cumulative likelihood is a stepping function, we should cut at the first step, which we do not (the algorithm is maximizing the likelihood over both Gaussians and the cut is most often than not in between the steps). A most important enhancement would be to model transitions from behavior to behavior, and predict to some extent the next behavior for a given time or identify critical events (weak signals) that would allow such predictions. Finally, we will apply this work to detect anomalies in a host and network intrusion detection system.

Acknowledgments. This work was supported by Akheros S.A.S./ANRT CIFRE grant #2014/0268, and the European project SCISSOR H2020-ICT-2014-1 #644425.

References

1. An, X., Jutla, D., Cercone, N.: Privacy intrusion detection using dynamic Bayesian networks. In: ACM International Conference Proceeding Series, vol. 156, pp. 208–215 (2006)
2. Beran, R.: Minimum hellinger distance estimates for parametric models. Ann. Stat. **5**, 445–463 (1977)
3. Casella, G., George, E.I.: Explaining the Gibbs sampler. Am. Stat. **46**(3), 167–174 (1992)
4. Charitos, T., Van Der Gaag, L.C., Visscher, S., Schurink, K.A., Lucas, P.J.: A dynamic Bayesian network for diagnosing ventilator-associated pneumonia in ICU patients. Expert Syst. Appl. **36**(2), 1249–1258 (2009)

5. Dean, T., Kanazawa, K.: A model for reasoning about persistence and causation. Comput. Intell. **5**(2), 142–150 (1989)
6. Gonzales, C., Dubuisson, S., Manfredotti, C.: A new algorithm for learning non-stationary dynamic Bayesian networks with application to event detection. In: The Twenty-Eighth International Flairs Conference (2015)
7. Grzegorczyk, M., Husmeier, D.: Non-stationary continuous dynamic Bayesian networks. In: Advances in Neural Information Processing Systems, pp. 682–690 (2009)
8. Grzegorczyk, M., Husmeier, D.: Non-homogeneous dynamic Bayesian networks for continuous data. Mach. Learn. **83**(3), 355–419 (2011)
9. Grzegorczyk, M., Husmeier, D., Edwards, K.D., Ghazal, P., Millar, A.J.: Modelling non-stationary gene regulatory processes with a non-homogeneous Bayesian network and the allocation sampler. Bioinformatics **24**(18), 2071–2078 (2008)
10. Kruegel, C., Mutz, D., Robertson, W., Valeur, F.: Bayesian event classification for intrusion detection. In: 2003 Proceedings of the 19th Annual Computer Security Applications Conference, pp. 14–23. IEEE (2003)
11. Lerner, U., Parr, R., Koller, D., Biswas, G., et al.: Bayesian fault detection and diagnosis in dynamic systems. In: AAAI/IAAI, pp. 531–537 (2000)
12. Mitra, V., Nam, H., Espy-Wilson, C.Y., Saltzman, E., Goldstein, L.: Gesture-based dynamic Bayesian network for noise robust speech recognition. In: 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5172–5175. IEEE (2011)
13. Murphy, K.P.: Dynamic Bayesian networks: representation, inference and learning. Ph.D. thesis, University of California, Berkeley (2002)
14. Mutz, D., Valeur, F., Vigna, G., Kruegel, C.: Anomalous system call detection. ACM Trans. Inf. Syst. Secur. (TISSEC) **9**(1), 61–93 (2006)
15. Ourston, D., Matzner, S., Stump, W., Hopkins, B.: Applications of hidden Markov models to detecting multi-stage network attacks. In: 2003 Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 10 p. IEEE (2003)
16. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo (2014)
17. Robinson, J.W., Hartemink, A.J.: Non-stationary dynamic Bayesian networks. In: Advances in Neural Information Processing Systems, pp. 1369–1376 (2009)
18. Robinson, J.W., Hartemink, A.J.: Learning non-stationary dynamic Bayesian networks. J. Mach. Learn. Res. **11**, 3647–3680 (2010)
19. Sicard, M., Baudrit, C., Leclerc-Perlat, M., Wuillemin, P.H., Perrot, N.: Expert knowledge integration to model complex food processes. Application on the camembert cheese ripening process. Expert Syst. Appl. **38**(9), 11804–11812 (2011)
20. Song, L., Kolar, M., Xing, E.P.: Time-varying dynamic Bayesian networks. In: Advances in Neural Information Processing Systems, pp. 1732–1740 (2009)
21. Xu, J., Shelton, C.R.: Continuous time Bayesian networks for host level network intrusion detection. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part II. LNCS (LNAI), vol. 5212, pp. 613–627. Springer, Heidelberg (2008)
22. Xu, J., Shelton, C.R.: Intrusion detection using continuous time Bayesian networks. J. Artif. Intell. Res. **39**, 745–774 (2010)
23. Yeung, D.Y., Ding, Y.: Host-based intrusion detection using dynamic and static behavioral models. Pattern Recogn. **36**(1), 229–243 (2003)
24. Zanero, S., Serazzi, G.: Unsupervised learning algorithms for intrusion detection. In: 2008. IEEE Network Operations and Management Symposium, NOMS 2008, pp. 1043–1048. IEEE (2008)