Learning and Selection of Dynamic Bayesian Networks for Non-Stationary Processes in Real Time

Matthieu Hourbracq,[†⋆] Pierre-Henri Wuillemin,[†]
Christophe Gonzales,[†] Philippe Baumard[⋆]
[⋆] Akheros S.A.S.
[†]Sorbonne Universités, UPMC Univ Paris 6, CNRS, UMR 7606 LIP6, Paris, France
matthieu.hourbracq@lip6.fr, pierre-henri.wuillemin@lip6.fr,
christophe.gonzales@lip6.fr, philippe.baumard@akheros.com

Abstract

Dynamic Bayesian Networks (DBNs) bring efficient tools to model complex multivariate dynamical systems learned from collected data and/or expert knowledge. Notwithstanding, the underlying generative Markov model is supposed homogeneous ; neither its topology nor its parameters evolve over time. Thus, learning a DBN to model a non-stationary process with this belief will lead to poor prediction capabilities. In order to account for non stationary processes, we build on a framework to identify transitions between underlying models and a framework to learn them in real time, without making hypothesis about their evolution. We present the tool performances on simulated datasets. Since we aim to use this to model and predict incongruities within an Intrusion Detection System (IDS) in near real-time, great care is ascribed to the capability to correctly detect transition times. Our prior results display the precision of our algorithm in the choice of transitions and therefore the quality of identified networks. At last we suggest future work.

Introduction

In numerous fields, especially information systems, physic and biology modeling, observed processes evolve over time on several scales, drawing complex trajectories. Correlations at any given time do not have to hold forever, and which entity influences another may change. Hence we cannot suppose stationarity if we wish to model such a process without being aware of the mechanism accountable for those changes, otherwise we get a behavior *averaged* over different ones.

Our aim is to model the behaviors of information systems in real time and in this setting programs interact with each other, modify their states and change their behavior accordingly. As such it seems realistic to assume the non-stationarity of the modeled system.

There are complex dynamical processes for which no complete deterministic model exists. In such cases, Dynamic Bayesian networks (Dean and Kanazawa 1989; Murphy 2002), which extend Bayesian networks (Pearl 2014), are a convenient formalism to describe those. The probabilistic and graphical essences of DBNs make them efficient tools to integrate both data and expert knowledge within a single representation. Yet it has limitations as the use of *dynamic* in DBN refers to the system evolving over time, not the dynamics of the network structure or its parameters. Indeed, once ascertained on a subset of observations, conditional dependencies and parameters are never revisited. It is unreasonable, in many applications, like our own, and even more so when data are not produced in a controlled manner, to assume homogeneity of the underlying model(s) describing which state the system is in. This issue has received attention in the last years giving rise to non-stationary dynamic Bayesian Networks (ns-DBN) (Robinson and Hartemink 2009; 2010; Grzegorczyk and Husmeier 2009; 2011; Gonzales, Dubuisson, and Manfredotti 2015) and time-varying dynamic Bayesian Networks (TV-DBN) (Song, Kolar, and Xing 2009), with some preliminary applications for system biology (Grzegorczyk et al. 2008).

Thereby we set our focus on non-stationary dynamic Bayesian networks.

The goal of this paper is to present a new algorithm to account for non-stationary processes in real time using non-stationary dynamic Bayesian networks. We begin by recalling preliminary notions of (d)BNs and ns-DBNs. Then we suggest a non-stationary learning algorithm and present our framework before evaluating its performances on a number of simulated cases to reveal strengths and weaknesses. At last, we conclude and expand on our future work.

Dynamic Bayesian Networks

DBNs extend Bayesian networks (Pearl 2014) with nodes $\{X_i(t), i = 1 \dots n\}$, representing random variables, indexed by time t. They provide a factored representation of the joint probability distribution $P(\mathbf{X}(1), \dots, \mathbf{X}(\tau))$ on a finite time interval $[1, \tau]$, encoding the beliefs about the trajectories of the dynamic process $\mathbf{X}(t)$:

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

where $\mathbf{U}_i(.)$ represents the set of parent nodes of $X_i(.)$ and $P\left(X_i\left(t\right) \mid \mathbf{U}_i\left(t\right)\right)$ 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.

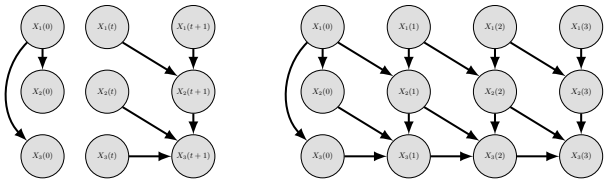

Figure 1: A 2-Time-Slice BN (2TBN) and its (unrolled) dynamic Bayesian network.

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 :

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

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

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

irst two time slices. We obtain a 2TBN such as in Figure 1.

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

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

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 fault detection (Lerner et al. 2000), speech recognition (Mitra et al. 2011), medical diagnosis (Charitos et al. 2009) or system biology (Sicard et al. 2011) but their applications on Intrusion Detection Systems are scarce (An, Jutla, and Cercone 2006). In this field Bayesian Networks are mainly used for classification purposes, as static models and deciding mechanism aggregating smaller models outputs, thus offering a summary of input data (Kruegel et al. 2003; Mutz et al. 2006). However, (Hidden) Markov Models have been extensively proposed to model system call traces and shell commands (Yeung and Ding 2003; Zanero and Serazzi 2008) as well as network data flow (Ourston et al. 2003).

Non-stationary dynamic Bayesian Networks

Ns-DBNs are represented as a collection of dynamic bayesians networks $\mathcal{B} : (\mathbf{\Theta}, \mathcal{G})$ organized by epochs of varying size (or transition times) $\mathcal{T} : \{(\mathcal{B}_m : (\mathbf{\Theta}, \mathcal{G})_m, \mathcal{T}_m)\}$. They leverage piece-wise stationarity over epochs to solve the issue with Equation 3. It is noteworthy that there is no framework to model the behavior of the transition times for ns-DBNs yet, which could also be non-stationary.

Learning of ns-DBNs amounts to the identification of epochs and their associated DBNs. Although different epochs may alter parameters, structures and even set of variables (see Figure 2), current learning algorithms focus on either structure or parameter evolution to cope with the size of the search space ; as such (Grzegorczyk and Husmeier 2009) focuses on parameters evolution with fixed structure whereas in (Robinson and Hartemink 2010) the focus is set on structural evolution, with edges gained and lost over time (non-stationarities from parameters only occur when the parameters of a conditional distribution imply a change of structure). For those later papers, the number of variables and their domains remain constant over time (even if some are not observed during whole epochs). They also are offline algorithms, requiring the availability of the whole database, and cannot be used in our online framework - except for our first learnings (when discovering new networks). Offline learning of ns-DBNs is achieved with the use of a modified DBNs scoring function accounting for the sufficient statistics, now specified by epoch, to find the best transitions. There is also an updated structural move set for learning the structure which also need to be specified by epoch. Still, we need simpler algorithms to achieve real time performance as we are streaming data. (Gonzales, Dubuisson, and Manfredotti 2015) is close to our approach allowing structure, parameters as well as variables and their domains to evolve over time, in real time. A test statistic is used to determine if the observed data is consistent with the model distribution, although new modalities always lead to new transitions. However, we use different criteria and a mechanism for windows overlapping events from different models to refine transition times.

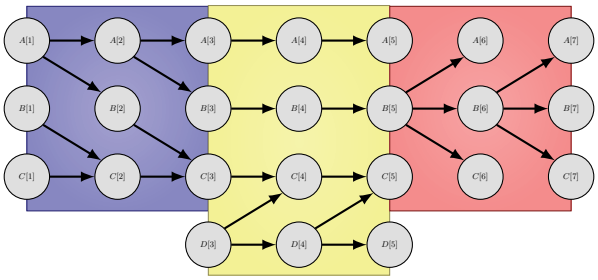

Figure 2: Non-Stationary dynamic Bayesian network (ns-DBN) with 3 different epochs. DBNs in different epochs may have different parameters, structures and/or variables.

Learning ns-DBNs

We present in this section a new framework to learn ns-DBNs in real time. We do not restrict ourselves to smooth evolutions from model to model as the assumption that two adjacent models are governed by similar distributions and/or similar structures is often made (Grzegorczyk and Husmeier 2009; Robinson and Hartemink 2010; Grzegorczyk and Husmeier 2011).

Data are streamed in real time, in a continuous manner using a sliding window $\mathbf{w}$. 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 $\mathbf{w}[\tau, \tau + r]$. It has to choose between using one of the known models or creating a new one. This choice is founded on the likelihood of the windowed data ; certainly the likelihood of a model will decrease if the underlying behavior changes (see Figure

3). The algorithm begins with a burn-in to get a network that serves as a starting point (we could and should use one of the offline algorithms mentioned previously to confirm there is only one model).

To confront a model m with $\mathbf{w}$ the algorithm uses the log-likelihood of the data in $\mathbf{w}$ against the network with structure $\mathcal{G}$ and parameters $\mathbf{\Theta}$:

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

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 $\mathbf{w}$.

This is computed for each known DBN. However we cannot select the best fitting model only by maximizing LL since having the best score does not ensure the corresponding network accurately represents the data ; the algorithm may also discover new models on the fly. As such, one can note that the distribution of the log-likelihood for a window $\mathbf{w}$ is approximately normally distributed (as the sum of $r+1$ i.i.d random variables, using the central limit theorem) assuming only one behavior is observed. Thus we 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 Figure 3). 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 :

$$m^* = \arg\max_{m \in \mathcal{M}_{0.97}} LL(\mathbf{w} : \mathbf{\Theta}_m, \mathcal{G}_m)$$
$$\text{with } \mathcal{M}_{0.97} = \left\{ m : \frac{LL_{tr}(\mathbf{\Theta}_m, \mathcal{G}_m)}{LL(\mathbf{w} : \mathbf{\Theta}_m, \mathcal{G}_m)} \geq 0.97 \right\} \tag{6}$$

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

When the algorithm predicts, given a current window, a model m_t different from the model m_{t-1} (i.e. m_t would be a newly created DBN or an already existing DBN), this prediction is not only about the change of behaviors but also about the time τ of this change ; the change point. In experiments, Figure 6 shows how a badly - fixed - sized window can mislead the learning with inaccurate change points. 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 a window with a change point at $c = 100\,000$. In order to estimate a correct value for c, we could rely on the change of slope in the cumulative of $P(\mathbf{X})$. To be more accurate, we choose to maximize, over c, the likelihood of a model where c separates two different Gaussian processes. With the optimized change point c^* we then update m_{t-1} on $\mathbf{w}[\tau - r, \tau - r + c^*]$ and update - or learn - m_t on $\mathbf{w}[\tau - r + c^*, \tau + r]$.

If the set of models in Equation 6 is empty, the algorithm will learn a new DBN from the window and select it. We use a non-informative Dirichlet prior, making the assumption that parameters and structure evolve without correlations from one model to another. We then use Gibbs sampling (Casella and George 1992) to produce a first estimate of LL_{tr}.

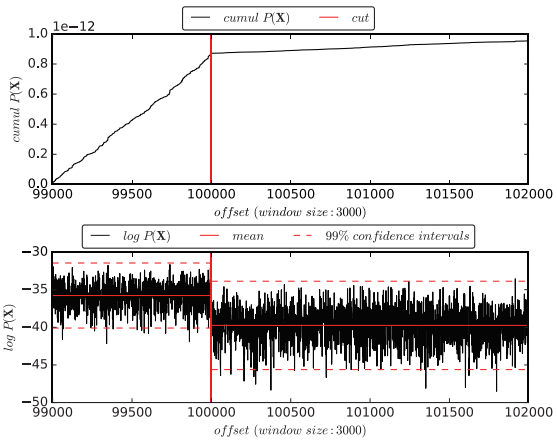

Figure 3: Cumulative $P(\mathbf{X})$ and $log\ P(\mathbf{X})$ for a window overlapping two different underlying models. The vertical red line is a transition between models, and horizontal red lines are Gaussian with 99% confidence interval.

If an existing model m is selected, its parameters and eventually its structure are updated with the new data. Indeed, as models get more observations, their structures will need to be reevaluated : at each order of magnitude, we re-estimate the network structures.

During the evolution of the non-stationary process, the set of observed variables and/or their modalities may change, as seen in Figure 2. This change does not have to always represent a change point, even more so as outliers are always present in real world data. If variable X_e is 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 X_e is in the database but not in $\mathcal{G}_m$, those informations are not exploitable for this model and 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 using the (non-informative) Dirichlet priors α_{ijk} parameters and then compute the likelihoods.

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.

Experiments and Results

Our experiment consists in modeling simulated non-stationary processes. The idea is to generate several networks, more or less close to each others, before sampling a database for each one and merging their content, using different epochs sizes to see the impact of sample size against network distance. To do so, we used the aGrUM library (http://agrum.lip6.fr), to generate a first DBN of 10 nodes

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

Table 1: Results for static windows, false negatives FN, false positives FP and true positives TP for transitions. For cuts, minimal, average and maximal error in events, with variance. For discovered networks, $precision$ and $recall$ over events.

by time-step of average domain size 7 ($[\![3, 10]\!]$) and average node degree 3. The structure and parameters of the model are perturbed using the Hellinger distance (Beran 1977) between all models as stopping criterion. We used multiple thresholds to see how far apart two networks have to be so they are recognized as two independent models . The use of different resolutions of the sliding window allows us 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 experiment with and without looking at a change point. 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 / frequency of transitions.

The (fictive) figure 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 (change points) 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).

We show in tables cuts average, minimal and maximal errors 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. $Recall$ is 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). Results were averaged for all thresholds of Hellinger distance due to pages restriction. We show a best case (Figure 5) and worst case (Figure 6, 7) scenario with and without looking for change points. The results for static and adaptive windows are presented in Tables 1 and 2, respectively.

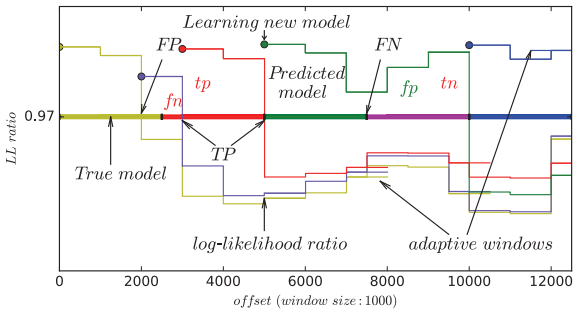

Figure 4: How to read figures.

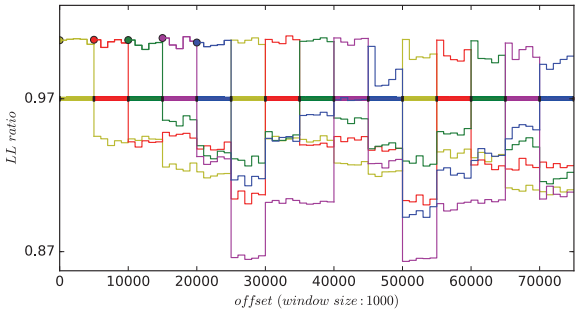

Figure 5: Results for epochs of $5K$ observations, $Hellinger < 0.8$, fixed window size

Static windows

When the epoch is a multiple of the window size the sliding window always contains observations from one model at a time. In such cases, correct transition times and models are always identified, with and without looking for a change point, as in Figure 5. However, errors arise when using arbitrary window sizes without looking for a change point as shown in Table 1 and Figures 6 and 7.

In Table 1, two issues explain the poor $precision$ and $recall$ for some experiments. The first issue arises when transitions were missed and some models start averaging several other true models, increasing noise and making further transitions harder and harder to detect, hence increasing FN of transitions and decreasing $precision$ and $recall$

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

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

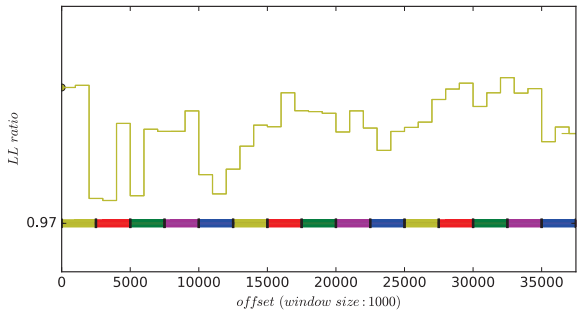

Figure 6: Results for epochs of $2K5$ observations, $Hellinger < 0.8$, fixed window size

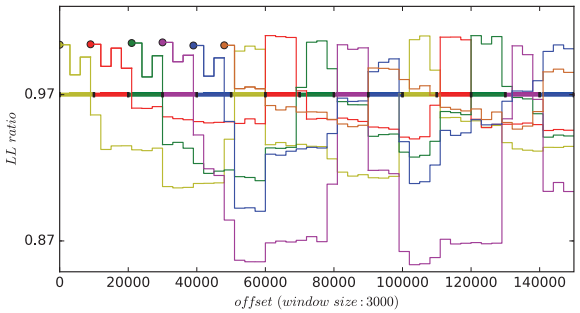

Figure 7: Results for epochs of $10K$ observations, $Hellinger < 0.8$, fixed window size

over events. We discovered fewer networks than we should have. Such a case is highlighted by Figure 6 and by the first two rows of Table 1, with the first row and figure 6 showing results for close true networks and the second row results for distinct true networks. The second issue arises when networks are made in excess as 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 Figure 7 (the brown network). We discovered more networks than we should have. Most of the time 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) at the same time.

Adaptive windows

Table 2 shows the results when looking for a change point and reveals that the size of the window has little effect on the correct identification of transitions and models, which should hold as long as the window size is lower than the epoch. Surprisingly, results are stable for small epochs given the domain size of the network. Both previous issues are solved by looking for a change point, as in Figures 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.

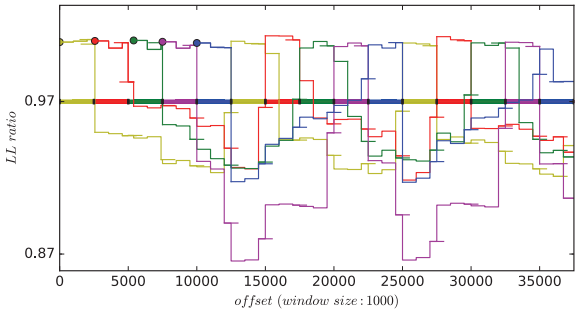

Figure 8: Results for epochs of $2K5$ observations, $Hellinger < 0.8$, dynamic window size

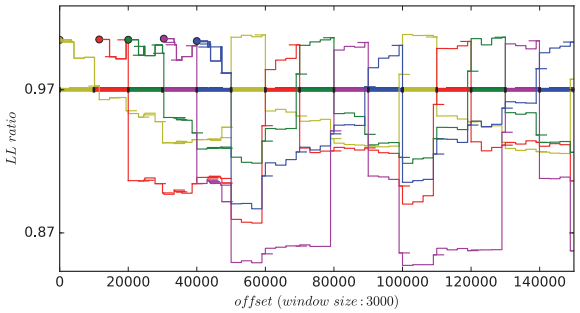

Figure 9: Results for epochs of $10K$ observations, $Hellinger < 0.8$, dynamic window size

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 and do not have to create change points and new models.

Conclusions and Future Work

We built a framework to learn and select Dynamic Bayesian Networks in a continuous manner as a representation of non-stationary processes. The framework is designed to be fast and accurate. Nonetheless, 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 models, 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 decreasing as the model is unobserved, the merging of models require to compare their joint probability distributions which involves heavy computations. A more robust change point detection algorithm could also be devised. The most important enhancement and our next work would be to model transitions from behavior to behavior, and predict to some extent the next one as well as key behaviors and critical events. Finally, we will apply this work to detect anomalies in a host and network intrusion detection system.

References

An, X.; Jutla, D.; and Cercone, N. 2006. Privacy intrusion detection using dynamic bayesian networks. In *ACM International Conference Proceeding Series*, volume 156, 208–215.

Beran, R. 1977. Minimum hellinger distance estimates for parametric models. *The Annals of Statistics* 445–463.

Casella, G., and George, E. I. 1992. Explaining the gibbs sampler. *The American Statistician* 46(3):167–174.

Charitos, T.; Van Der Gaag, L. C.; Visscher, S.; Schurink, K. A.; and Lucas, P. J. 2009. A dynamic bayesian network for diagnosing ventilator-associated pneumonia in icu patients. *Expert Systems with Applications* 36(2):1249–1258.

Dean, T., and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Computational intelligence* 5(2):142–150.

Gonzales, C.; Dubuisson, S.; and Manfredotti, C. 2015. A new algorithm for learning non-stationary dynamic bayesian networks with application to event detection. In *The Twenty-Eighth International Flairs Conference*.

Grzegorczyk, M., and Husmeier, D. 2009. Non-stationary continuous dynamic bayesian networks. In *Advances in Neural Information Processing Systems*, 682–690.

Grzegorczyk, M., and Husmeier, D. 2011. Non-homogeneous dynamic bayesian networks for continuous data. *Machine Learning* 83(3):355–419.

Grzegorczyk, M.; Husmeier, D.; Edwards, K. D.; Ghazal, P.; and Millar, A. J. 2008. Modelling non-stationary gene regulatory processes with a non-homogeneous bayesian network and the allocation sampler. *Bioinformatics* 24(18):2071–2078.

Kruegel, C.; Mutz, D.; Robertson, W.; and Valeur, F. 2003. Bayesian event classification for intrusion detection. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, 14–23. IEEE.

Lerner, U.; Parr, R.; Koller, D.; Biswas, G.; et al. 2000. Bayesian fault detection and diagnosis in dynamic systems. In *AAAI/IAAI*, 531–537.

Mitra, V.; Nam, H.; Espy-Wilson, C. Y.; Saltzman, E.; and Goldstein, L. 2011. Gesture-based dynamic bayesian network for noise robust speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, 5172–5175. IEEE.

Murphy, K. P. 2002. *Dynamic bayesian networks: representation, inference and learning*. Ph.D. Dissertation, University of California, Berkeley.

Mutz, D.; Valeur, F.; Vigna, G.; and Kruegel, C. 2006. Anomalous system call detection. *ACM Transactions on Information and System Security (TISSEC)* 9(1):61–93.

Ourston, D.; Matzner, S.; Stump, W.; and Hopkins, B. 2003. Applications of hidden markov models to detecting multi-stage network attacks. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, 10–pp. IEEE.

Pearl, J. 2014. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Robinson, J. W., and Hartemink, A. J. 2009. Non-stationary dynamic bayesian networks. In *Advances in Neural Information Processing Systems*, 1369–1376.

Robinson, J. W., and Hartemink, A. J. 2010. Learning non-stationary dynamic bayesian networks. *The Journal of Machine Learning Research* 11:3647–3680.

Sicard, M.; Baudrit, C.; Leclerc-Perlat, M.; Wuillemin, P.-H.; and Perrot, N. 2011. Expert knowledge integration to model complex food processes. application on the camembert cheese ripening process. *Expert Systems with Applications* 38(9):11804–11812.

Song, L.; Kolar, M.; and Xing, E. P. 2009. Time-varying dynamic bayesian networks. In *Advances in Neural Information Processing Systems*, 1732–1740.

Yeung, D.-Y., and Ding, Y. 2003. Host-based intrusion detection using dynamic and static behavioral models. *Pattern recognition* 36(1):229–243.

Zanero, S., and Serazzi, G. 2008. Unsupervised learning algorithms for intrusion detection. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, 1043–1048. IEEE.