A New Algorithm for Learning Non-Stationary Dynamic Bayesian Networks: Application to Event Detection

Christophe Gonzales[1], **Séverine Dubuisson**[2], **Cristina Manfredotti**[1]
[1] Sorbonne Universites, UPMC Univ Paris 06, UMR 7606, LIP6, Paris, France.
[2] Sorbonne Universites, UPMC Univ Paris 06, UMR 7222, ISIR, Paris, France.

Abstract

Dynamic Bayesian networks (DBN) are a popular framework for managing uncertainty in time-evolving systems. Their efficient learning has thus received many contributions in the literature. But, most often, those assume that the data to be modeled by a DBN are generated by a stationary process, i.e., neither the structure nor the parameters of the BNs evolve over time. Unfortunately, there exist real-world problems where such a hypothesis is highly unrealistic, e.g., in video event recognition, social networks or road traffic analysis. In this paper, we propose a principled approach to learn the structure and parameters of "non-stationary DBNs", that can cope with such situations. Our algorithm is specifically designed to work in situations where all input data are streamed. Unlike previous works on non-stationary DBN learning, we make no restrictive assumption about the way the structure evolves or over parameters' independence during this evolution. Yet, as highlighted in our experimentations, our algorithm scales very well. Its lack of restrictive assumptions makes it very effective to detect events (evolutions), which is confirmed by our experimentations.

Introduction

Over the last 25 years, Bayesian networks (BN) have become the reference framework for reasoning with uncertainty (**?**). Their popularity stimulated the need for various extensions, e.g., probabilistic relational models for dealing with very high-dimensional problems (**?**), but also temporal extensions to deal with dynamic systems like DBNs (**?**). Over the years, the latter have become very popular. Basically, a DBN is a pair of Bayesian networks, representing the uncertainty over the random variables in the first time slice and over those in pairs of consecutive time slices respectively. Appending $(n-1)$ times the second BN to the first one is a simple and effective way to create a "grounded" BN defined over an horizon of n time slices. By its definition, learning the structure and parameters of a DBN consists essentially of applying separately classical learning algorithms on its two BNs, a task for which many efficient algorithms do exist (**?**; **?**; **?**; **?**).

However, in many practical situations, it is unreasonable to assume that neither the structure nor the parameters of the DBN evolve over time. As a simple example, consider tracking two individuals in a video who first walk together and split after a few moments. As their trajectories differ at the split point, using the same DBN to model their behavior before and after the split would clearly result in poor tracking. The same reasoning could also be applied to neural patterns that evolve during life, road traffic for which rush hours' densities are different from those of the rest of the day, and to many other domains. Non-stationary DBNs (nsDBN) have been introduced precisely to cope with this time-evolving dimension (**?**; **?**). Essentially, they consist of modeling the dynamic system using a set of pairs (BN,time interval). Similarly to classical DBNs, appending all these BNs on the time intervals on which they are defined results in a "grounded" BN defined over the union of all these intervals.

While their definition is quite simple, there exist surprisingly few algorithms for learning nsDBNs in the literature, see, e.g., (**?**; **?**; **?**). But all these methods have serious shortcomings. For instance, it is assumed in (**?**) that only parameters can change, the structure being necessarily fixed. In (**?**; **?**), the evolution of the structure is supposed to follow a given distribution. Such a hypothesis is questionable: actually, removing an arc representing a strong dependence between two random variables is considered equivalent to removing one that represents a very weak dependence. In addition, in these papers, the data over all the time slices need be known prior to the nsDBN structure learning, which inevitably rules out applications where input data are streamed (like in videos). In (**?**), data are supposed to be constituted by only one observation per time slice, which requires that the generating process remains stationary for long periods. In (**?**), random variables need be continuous, hence ruling out applications where data are discrete. In (**?**), when the structure evolves, the parameters before and after the evolution are supposed to be independent. This hypothesis may not seem so realistic: why should a structure modification around a node X_n make the conditional probabilities of all the other nodes after this modification be independent of those before the change? Due to these strong requirements, the above methods are not very well suited for situations where data are either discrete or are streamed. In this paper, we propose a new learning algorithm devoted to those situations. In particular, it is designed to work in a streaming mode and to enable parameters' dependencies between different time intervals.

The paper is organized as follows. In the first section, we recall the basics of nsDBNs. Then, we present our method and, in the third section, we demonstrate its efficiency, both in terms of the quality of the nsDBN found and in terms of computation times. Finally, a conclusion is provided.

Non-Stationary DBNs and Their Learning

In all the paper, bold letters refer to sets and random variables are all considered to be discrete. A Bayesian network is a compact probabilistic graphical model defined as:

Definition 1 *A BN is a pair $(\mathcal{G}, \mathbf{\Theta})$ where $\mathcal{G} = (\mathbf{V}, \mathbf{A})$ is a directed acyclic graph (DAG), $\mathbf{V}$ represents a set of random variables*[1]*, $\mathbf{A}$ is a set of arcs, and $\mathbf{\Theta} = \{P(X|\mathbf{Pa}(X))\}_{X\in\mathbf{V}}$ is the set of the conditional probability distributions (CPT) of the nodes / random variables X in $\mathcal{G}$ given their parents $\mathbf{Pa}(X)$ in $\mathcal{G}$. The BN encodes the joint probability over $\mathbf{V}$ as:*

$$P(\mathbf{V}) = \prod_{X\in\mathbf{V}} P(X|\mathbf{Pa}(X)). \tag{1}$$

nsDBNs are a temporal generalization of BNs:

Definition 2 *A nsDBN is a sequence of pairs $\langle(\mathcal{B}_h, T_h)\rangle_{h=0}^{m}$, where $T_0 = 0$ represents the first time slice, $\mathcal{B}_0$ is the BN representing the distribution over the random variables at time 0, and each $\mathcal{B}_h$, $h = 1, \dots, m$, is a BN representing the conditional probability of the random variables at time t given those at time $t-1$, for all t in time intervals $\mathbf{E}_h = \{T_{h-1}+1, \dots, T_h\}$. T_h and $\mathbf{E}_h$ are called a* transition time *and an* epoch *respectively. By convention, $\mathbf{E}_0 = \{0\}$.*

A DBN is a nsDBN in which the sequence contains only two pairs. Figure 1 shows a nsDBN (left) and its grounded BN (right). Note that, by Definition 2, nsDBNs are defined over discrete time horizons. Learning a nsDBN consists in i) determining the set of transition times T_h at which the network evolves and ii) learning the structure and parameters of each $\mathcal{B}_h$ (as in a classical DBN learning). Usually, the structure is learnt first and, given it, parameters are estimated by maximum likelihood or maximum a posteriori.

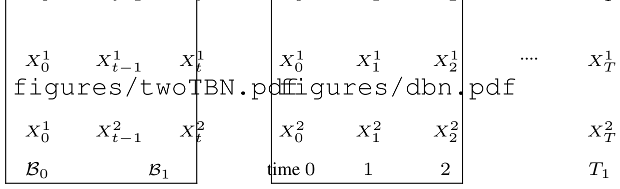

Figure 1: A nsDBN $\langle(\mathcal{B}_0, 0), (\mathcal{B}_1, T_1)\rangle$ (left, shaded) and its ground BN (right).

[1]By abuse of notation, we use interchangeably $X \in \mathbf{V}$ to denote a node in the BN and its corresponding random variable.

In the Bayesian framework, learning the structure of a single BN $\mathcal{B} = (\mathcal{G}, \mathbf{\Theta})$ from a database $\mathbf{D}$ is equivalent to computing $\mathcal{G} = \text{Argmax}_{G\in\mathbb{G}} P(G|\mathbf{D}) = \text{Argmax}_{G\in\mathbb{G}} P(G, \mathbf{D})$, where $\mathbb{G}$ is the set of all possible DAGs, which is equivalent to optimizing the following integral:

$$P(G, \mathbf{D}) = P(G) \int_{\boldsymbol{\theta}} P(\mathbf{D}|G, \boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{2}$$

where $\boldsymbol{\theta}$ is the set of parameters of the BN, and $\pi(\boldsymbol{\theta})$ and $P(G)$ are priors over the set of parameters and structures respectively. Assuming i) Dirichlet priors on parameters $\boldsymbol{\theta}$, ii) independence of parameters for different values of $\mathbf{Pa}(X)$ in CPT $P(X|\mathbf{Pa}(X))$, iii) a complete database $\mathbf{D}$ (i.e., there are no missing values), iv) data in $\mathbf{D}$ are i.i.d., it is shown in (**?**) that Equation (2) is equal to the Bayesian Dirichlet (BD) score:

$$P(G) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(N_{ij} + \alpha_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk} + \alpha_{ijk})}{\Gamma(\alpha_{ijk})}, \tag{3}$$

where Γ is the usual Gamma function, n represents the number of random variables in the BN, r_i and q_i are the domain sizes of the ith random variable X_i and of its parents $\mathbf{Pa}(X_i)$ respectively; N_{ijk} represents the number of records in the database in which the kth value of X_i and the jth value of its parents were observed. Similarly, $N_{ij} = \sum_k N_{ijk}$ is the number of records in which the jth value of $\mathbf{Pa}(X_i)$ was observed. α_{ijk} represents the a priori that X_i and its parents are equal to their kth and jth values respectively. Finally, $\alpha_{ij} = \sum_k \alpha_{ijk}$. Usually, the prior $P(G)$ on structures is assumed to be uniform, hence $P(G)$ is often not taken into account in Equation (3).

For nsDBNs, Equation (3) is adapted in (**?**) as:

$$\begin{aligned} P(G_0, \dots, G_m) \prod_{h=0}^{m} \prod_{i=1}^{n} \prod_{j=1}^{q_i} & \frac{\Gamma(\alpha_{ij}(\mathbf{E}_h))}{\Gamma(N_{ij}(\mathbf{E}_h) + \alpha_{ij}(\mathbf{E}_h))} \\ \times \prod_{k=1}^{r_i} & \frac{\Gamma(N_{ijk}(\mathbf{E}_h) + \alpha_{ijk}(\mathbf{E}_h))}{\Gamma(\alpha_{ijk}(\mathbf{E}_h))}, \end{aligned} \tag{4}$$

where the G_h's are the structures at each epoch, and the quantities in the Gamma functions are similar to those in (3) except that they are computed on subdatabases of $\mathbf{D}$ corresponding to epochs $\mathbf{E}_h$. In order to derive this equation, (**?**) assume that the CPT's parameters in each epoch are *mutually independent* of those in the other epochs. It also assumed that the domains of the random variables do not change over time, which may not always be realistic. For instance, in video tracking, the positions of the tracked object change over time and it is suboptimal to consider at each epoch the union of all the possible positions over all the frames. Finally, to keep the search on the structures tractable, (**?**) assumes that the evolution of the structure over time follows a truncated exponential. This enables to substitute $P(G_0, \dots, G_m)$ by $e^{-\lambda_s s}$, where s is the number of graph transformations used sequentially to create all the G_h's from the G_{h-1}'s. Again, as discussed in the introduction, this assumption is questionable. In the next section, we propose another learning algorithm that overcomes these problems.

A New Learning Algorithm

In this section, we propose a new algorithm designed for streaming problems (for example, for real-time event detection in video sequences). We thus assume that database $\mathbf{D}$ is observed sequentially, one time slice at a time. Let $\mathbf{D}_t$ be the data observed precisely at time t. Assume that somehow we determined that $\mathcal{B}_h = (\mathcal{G}_h, \mathbf{\Theta}_h)$ was the best BN modeling the uncertainties up to time T_h and, now, at time $t = T_h + 1$, we observe new data $\mathbf{D}_t$ that are likely to have been generated by another BN $\mathcal{B}_{h+1} = (\mathcal{G}_{h+1}, \mathbf{\Theta}_{h+1})$. Following the preceding section, and taking into account knowledge at time T_h, i.e., taking into account BN $\mathcal{B}_h$, graph $\mathcal{G}_{h+1}$ can be determined as $\mathcal{G}_{h+1} = \text{Argmax}_{G\in\mathbb{G}} P(G|\mathbf{D}_t, \mathcal{B}_h) = \text{Argmax}_{G\in\mathbb{G}} P(G, \mathbf{D}_t|\mathcal{G}_h, \mathbf{\Theta}_h)$.

$$\begin{aligned}
P(G, \mathbf{D}_t|\mathcal{G}_h, \mathbf{\Theta}_h) &= \int_{\boldsymbol{\theta}_{h+1}} P(G, \boldsymbol{\theta}_{h+1}, \mathbf{D}_t|\mathcal{G}_h, \mathbf{\Theta}_h) d\boldsymbol{\theta}_{h+1} \\
&= \int_{\boldsymbol{\theta}_{h+1}} P(\mathbf{D}_t|G, \boldsymbol{\theta}_{h+1}, \mathcal{G}_h, \mathbf{\Theta}_h)\pi(\boldsymbol{\theta}_{h+1}|G, \mathcal{G}_h, \mathbf{\Theta}_h) \\
&\qquad\qquad \times P(G|\mathcal{G}_h, \mathbf{\Theta}_h) d\boldsymbol{\theta}_{h+1} \\
&= P(G|\mathcal{G}_h, \mathbf{\Theta}_h) \int_{\boldsymbol{\theta}_{h+1}} P(\mathbf{D}_t|G, \boldsymbol{\theta}_{h+1}, \mathcal{G}_h, \mathbf{\Theta}_h) \\
&\qquad\qquad \times \pi(\boldsymbol{\theta}_{h+1}|G, \mathcal{G}_h, \mathbf{\Theta}_h) d\boldsymbol{\theta}_{h+1}
\end{aligned}$$

It is reasonable to assume that, given $(G, \boldsymbol{\theta}_{h+1})$, i.e., given the BN at time t, $\mathbf{D}_t$ is independent of the BN at time $T_h < t$, i.e., it is independent of $(\mathcal{G}_h, \mathbf{\Theta}_h)$. Therefore, the above equation is equivalent to:

$$\begin{aligned}
P(G, \mathbf{D}_t|\mathcal{G}_h, \mathbf{\Theta}_h) &= P(G|\mathcal{G}_h, \mathbf{\Theta}_h) \\
\times &\int_{\boldsymbol{\theta}_{h+1}} P(\mathbf{D}_t|G, \boldsymbol{\theta}_{h+1})\pi(\boldsymbol{\theta}_{h+1}|G, \mathcal{G}_h, \mathbf{\Theta}_h) d\boldsymbol{\theta}_{h+1}.
\end{aligned} \tag{5}$$

Now, let us review all the terms in the above equation. $\pi(\boldsymbol{\theta}_{h+1}|G, \mathcal{G}_h, \mathbf{\Theta}_h)$ is the prior over the parameters of the BN at time t. A classical assumption, notably justified in (**?**), is that this prior is a Dirichlet distribution. We also make this assumption. Doing so and assuming on subdatabase $\mathbf{D}_t$ hypotheses i) to iv) provided in the preceding section for learning static BNs will obviously results in the above integral being equal to the product of Gamma functions of Equation (3). As pointed-out in (**?**), a difficulty with using the α_{ijk}'s is that the user must specify those *a priori* over all the parameter space. To avoid this problem, people usually add other assumptions that enforce specific values for the α_{ijk}'s. For instance, the K2 and BDeu scores enforce $\alpha_{ijk} = 1$ and $\alpha_{ijk} = \frac{N'}{r_i q_i}$ (with N' an effective sample size) respectively. In our framework, the α_{ijk}'s correspond to $\pi(\boldsymbol{\theta}_{h+1}|G, \mathcal{G}_h, \mathbf{\Theta}_h)$, i.e., they are the *a priori* over parameters $\boldsymbol{\theta}_{h+1}$ of $\mathcal{B}_{h+1}$ given the BN $\mathcal{B}_h$ at time T_h. Therefore, if $\hat{\mathcal{B}} = (G, \hat{\mathbf{\Theta}})$ is, among all the BNs whose structures are G, that which minimizes the Kullback-Leibler distance with $\mathcal{B}_h$, it makes sense to assume that $\pi(\boldsymbol{\theta}_{h+1}|G, \mathcal{G}_h, \mathbf{\Theta}_h)$ is a Dirichlet distribution with hyperparameters $N'\hat{\mathbf{\Theta}}$. Actually, if N' is set to $+\infty$, the Dirichlet distribution will tend as much as possible toward a Dirac around the distribution of $\mathcal{B}_h$ (given the new structure G). If N' is set to a small value, the impact of $\mathcal{B}_h$ over $\boldsymbol{\theta}_{h+1}$ will be negligible. By adjusting appropriately the value of N', the impact of $\mathcal{B}_h$ over $\boldsymbol{\theta}_{h+1}$ can thus be precisely controlled.

Lemma 1 *Let $\hat{\mathcal{B}} = (G, \hat{\mathbf{\Theta}})$ be the BN with structure G whose Kullback-Leibler distance with $\mathcal{B}_h$ is minimal. Let $P_{\mathcal{B}_h}$ and $P_{\hat{\mathcal{B}}}$ be the probability distributions encoded by $\mathcal{B}_h$ and $\hat{\mathcal{B}}$ respectively. Finally, for any node X in G, let $\mathbf{Pa}_G(X)$ denote the set of parents of X in G. Then $\hat{\mathbf{\Theta}}$ is defined as, for any X, $P_{\hat{\mathcal{B}}}(X|\mathbf{Pa}_G(X)) = P_{\mathcal{B}_h}(X|\mathbf{Pa}_G(X))$.*

Computing $P_{\hat{\mathcal{B}}}(X|\mathbf{Pa}_G(X))$ is thus just a matter of inference in BN $\mathcal{B}_h$. If this inference is computationally difficult due to a too strong structural difference between G and $\mathcal{G}_h$ (for instance, if, when applying a junction tree-based algorithm, the treewidth of the latter is too high), $P_{\hat{\mathcal{B}}}(X|\mathbf{Pa}_G(X))$ can still be approximated quickly and efficiently by maximum likelihood exploiting the subdatabase at epoch $\mathbf{E}_h$.

Let us now examine the first term in Equation (5), i.e., $P(G|\mathcal{G}_h, \mathbf{\Theta}_h)$. This is the prior over the structures' space. In (**?**), G is assumed to be independent of $\mathbf{\Theta}_h$ given $\mathcal{G}_h$ and the prior is thus equal to $P(G|\mathcal{G}_h)$; the latter is modeled as a truncated exponential that penalizes graphs G that are too different from $\mathcal{G}_h$. As pointed out in the introduction, this does not take into account the strength of the arcs in the network, which are determined by factor $\mathbf{\Theta}_h$. Yet, arcs representing weak dependences are certainly more likely to be deleted than those representing very strong dependences. In our framework, we propose to take into account this factor. There exist several criteria to measure the strength of an arc. In (**?**), the authors propose to exploit the conditional mutual information criterion, leading to assessing the strength of arc $X \to Y$ as:

$$I(X, Y|\mathbf{Z}) = \sum_{X,\mathbf{Z}} P(X, \mathbf{Z}) \sum_Y P(Y|X, \mathbf{Z}) \log \frac{P(Y|X, \mathbf{Z})}{P(Y|\mathbf{Z})},$$

where $\mathbf{Z} = \mathbf{Pa}(Y)\backslash\{X\}$. (**?**) suggest to approximate this equation as follows to speed-up its computation:

$$\hat{I}(X, Y|\mathbf{Z}) = \sum_{X,\mathbf{Z}} P(X)P(\mathbf{Z}) \sum_Y P(Y|X, \mathbf{Z}) \log \frac{P(Y|X, \mathbf{Z})}{P(Y|\mathbf{Z})}.$$

We can now extend the truncated exponential used by (**?**) as follows: let $\langle(X_{(s)}, Y_{(s)}, A_s)\rangle_{s=1}^c$ be a sequence of transformations ($A_s \in$ {arc addition (add), arc deletion (del), arc reversal (rev)}) changing $\mathcal{G}_h$ into G. We propose to score each graph G as:

$$P(G) \propto \hat{P}(G) = \prod_{s=1}^{c} e^{f(X_{(s)}, Y_{(s)}, A_s)}, \tag{6}$$

where $f(X_{(s)}, Y_{(s)}, A_s)$ is equal to:

$$\begin{cases} -\lambda_d \hat{I}(X_{(s)}, Y_{(s)}|\mathbf{Pa}(Y_{(s)})\backslash\{X_{(s)}\}) & \text{if } A_s = \text{del} \\ -\lambda_a / \hat{I}(X_{(s)}, Y_{(s)}|\mathbf{Pa}(Y_{(s)})) & \text{if } A_s = \text{add} \\ \frac{1}{2}(f(X_{(s)}, Y_{(s)}, \text{del}) + f(Y_{(s)}, X_{(s)}, \text{add})) & \text{if } A_s = \text{rev} \end{cases}$$

and λ_d and λ_a are two positive constants that represent our knowledge about the possible amount of differences between $\mathcal{G}_h$ and G. Conditional mutual informations being always non-negative, this formula tends to penalize graph modifications while taking into account the strengths of the arcs. As for the prior on the parameters, computing $\hat{P}(G)$ is thus just a matter of inference in BN $\mathcal{B}_h$. In our framework, $P(G, \mathbf{D}_t|\mathcal{G}_h, \mathbf{\Theta}_h)$ is thus equal to:

$$\prod_{s=1}^{c} e^{f(X_{(s)},Y_{(s)},A_s)} \prod_{i=1}^{n}\prod_{j=1}^{q_i} \frac{\Gamma(N'\hat{\boldsymbol{\theta}}_{ij}(\mathbf{E}_h))}{\Gamma(N_{ij}(t)+N'\hat{\boldsymbol{\theta}}_{ij}(\mathbf{E}_h))} \times \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk}(t)+N'\hat{\boldsymbol{\theta}}_{ijk}(\mathbf{E}_h))}{\Gamma(N'\hat{\boldsymbol{\theta}}_{ijk}(\mathbf{E}_h))}, \tag{7}$$

where N' is an effective sample size, $\hat{\boldsymbol{\theta}}_{ijk}(\mathbf{E}_h)$ is the probability according to $\hat{\mathcal{B}}$ of Lemma 1 that X_i is equal to its kth value given the jth configuration of its parents, and $\hat{\boldsymbol{\theta}}_{ij}(\mathbf{E}_h) = \sum_{k=1}^{r_i} \hat{\boldsymbol{\theta}}_{ijk}(\mathbf{E}_h)$. $N_{ijk}(t)$ represents the number of observations in $\mathbf{D}_t$ of the kth value of X_i and the jth configuration of its parents.

To complete the algorithm, there remains to determine the transition times T_h. In (**?**), it is suggested that the search space for the learning algorithm should be the structures' space times the set of transitions' space. By adding operators that change the number of transitions as well as operators that change the transition time's locations. This increases considerably the size of the search space, making the convergence of the learning algorithm difficult to reach.

In (**?**), the authors propose to compute, for each record $d = (x_1, \ldots, x_n)$ of observations of all the variables X_i in subdatabase $\mathbf{D}_{\mathbf{E}_h} \cup \mathbf{D}_t$ corresponding to time slices $\mathbf{E}_h \cup \{t\}$, and each node X_i, a quantity $c_d = P_{\mathcal{B}_h}(X_i = x_i)/P_{\mathcal{B}_h}(X_i = x_i|X_j = x_j, j \neq i)$. For a given node X_i, the set of c_i's thus constructed form a vector of length, say v, from which a quantity $C_i = \sum_{d=1}^{v} c_d \cos(\frac{\pi(d+0.5)}{v})$ is computed. C_i corresponds to the negative of the second component of a discrete cosine transform. The authors argue that a high value of C_i indicates a transition time. While this formula, notably the use of c_d, is not really mathematically justified, it has a major advantage over the method in (**?**): it does not increase the search space, hence enabling fast learning algorithms.

In our framework, we propose yet another method, which both limits the size of the search space and is mathematically well justified. We suggest to perform classical goodness-of-fit tests of BN $\mathcal{B}_h$ with database $\mathbf{D}_t$. More precisely, for each node X_i of the BN, we compute the χ^2 statistics over the subset of $\mathbf{D}_t$ corresponding to $\{X_i\} \cup \mathbf{Pa}(X_i)$, where $\mathbf{Pa}(X_i)$ is the set of parents of X_i in $\mathcal{B}_h$. If at least one of those statistics indicate that it is more likely that $\mathbf{D}_t$ has been generated by a BN different from $\mathcal{B}_h$ than by $\mathcal{B}_h$ itself, then this means that we have reached a new transition time, else t should not be considered as a transition time. Prior to using the goodness-of-fit tests, if $\mathbf{D}_t$ does not contain the same variables as $\mathcal{B}_h$ (for instance, new random variables have been added or others have been discarded), or if some observed values do not match the domains of their random variables in $\mathcal{B}_h$, our algorithm considers that a new transition time has been reached. Note that, in streaming applications, it is quite possible to observe some random variable's values only after a given time. For instance, in video tracking, a person might be on the left of the scene at the beginning of the video and on the right only at the end, and it would be suboptimal from a learning perspective to consider that random variable "location of the person" is defined at every time slice over all the possible values from left to right (because this would increase drastically the domain size of the random variable, hence increasing the noise in the BD score used for structure learning).

Our complete algorithm is given in Algo. 1. Note that, when t is not considered as a transition time, i.e., when $\mathbf{D}_t$ is likely to have been generated by $\mathcal{B}_h$, we shall exploit this database to refine the parameters of $\mathcal{B}_h$, e.g., using the algorithm advocated in (**?**). We will show in the next section the efficiency and effectiveness of our algorithm.

```
learn $\mathcal{B}_0$ from database $\mathbf{D}_0$
learn $\mathcal{B}_1$ from database $\mathbf{D}_1$
$h \leftarrow 1$
while a new database $\mathbf{D}_t$ is observed do
    if the variables in $\mathbf{D}_t$ or their values differ from those in $\mathcal{B}_h$, or if a goodness-of-fit test indicates a transition time then
        learn $\mathcal{B}_{h+1}$ using Equation (7)
        $h \leftarrow h + 1$
    else
        update the parameters in $\mathcal{B}_h$ using $\mathbf{D}_t$
```

Algorithm 1: nsDBN Learning.

Experimentations

For our experimentations, we generated nsDBNs as follows: for $\mathcal{B}_0$, we started from two well-known BNs, "asia" and "alarm", with 8 and 37 nodes respectively, in which we perturbed the CPTs randomly. Then, using the random generator proposed in (**?**), we generated the other $\mathcal{B}_h$'s for 4 epochs ($h = 1, \ldots, 4$). Again, the CPTs of all the $\mathcal{B}_h$'s were randomly perturbed. Then we sampled from these nsDBNs to create databases $\mathbf{D}_t$. We performed essentially two sets of experiments. In the first one, each epoch was set to 10 time slices and databases were generated with different numbers of records (20, 100, 250, 500, 750, 1000, 2500, 5000) whereas, in the second set, the number of records was fixed to either 500 or 2000 but the number of time slices in the epochs varied from 5 to 30 with a step of 5. For each of those configurations, 30 experiments were performed, resulting in an overall of 1200 different nsDBNs. Tables 1 to 3 average over the 30 experiments the results obtained by our algorithm on these nsBNs on a PC with an Intel Xeon E5 2630 at 2.60GHz running on Linux. In all the experimentations, our algorithm's χ^2 test significance level α was set to 0.01.

In Table 1, columns TP, FN, FP refer to the number of "true positives", "false negatives" and "false positives" w.r.t. the transition times. As can be seen, even with moderate-size databases ($\sim$250 records per time slice), our algorithm finds efficiently the transitions of the alarm-based nsDBNs

Table 1: . Experimentations with fixed-size epochs (10 time slices) and databases of varying sizes

Network	Size	KL mean	KL std	KL min	KL max	TP	FN	FP	Time
Alarm	20	21.19±3.80	15.52±1.47	6.31±4.14	39.47±2.58	1.17±0.37	3.83±0.37	0.27±0.51	2.65±0.97
	100	16.10±3.13	5.51±1.51	7.71±5.04	23.43±2.71	3.43±1.05	1.57±1.05	1.57±1.05	9.34±2.81
	250	13.36±1.69	7.33±0.89	2.01±2.19	23.04±2.13	4.93±0.25	0.07±0.25	1.40±0.99	14.27±1.53
	500	11.65±1.13	6.89±0.61	2.26±1.06	22.91±1.85	5.00±0.00	0.00±0.00	0.80±1.08	15.75±1.20
	750	12.25±1.08	6.20±0.58	3.55±0.55	23.07±1.07	5.00±0.00	0.00±0.00	1.03±0.84	17.24±1.26
	1000	12.63±1.12	6.25±0.49	2.88±0.29	22.45±1.05	5.00±0.00	0.00±0.00	1.13±1.28	19.57±1.18
	2500	10.49±1.04	6.38±0.59	2.10±0.12	20.70±1.35	5.00±0.00	0.00±0.00	1.73±1.63	27.89±2.50
	5000	6.96±1.10	3.38±0.78	1.09±0.22	12.45±2.06	5.00±0.00	0.00±0.00	3.03±1.74	39.02±3.46
Asia	20	3.85±0.82	0.86±0.77	2.98±0.81	4.92±1.62	1.37±0.48	3.63±0.48	1.00±0.63	0.47±0.16
	100	1.20±0.16	0.48±0.15	0.50±0.15	2.03±0.42	4.67±0.60	0.33±0.60	1.13±1.02	2.06±0.46
	250	0.75±0.15	0.35±0.12	0.27±0.06	1.46±0.40	5.00±0.00	0.00±0.00	1.10±1.04	4.13±0.67
	500	0.58±0.14	0.36±0.11	0.13±0.06	1.32±0.35	5.00±0.00	0.00±0.00	1.43±1.15	4.65±0.47
	750	0.53±0.14	0.36±0.11	0.10±0.06	1.30±0.35	5.00±0.00	0.00±0.00	1.77±1.71	4.96±0.76
	1000	0.51±0.13	0.36±0.10	0.09±0.05	1.27±0.34	5.00±0.00	0.00±0.00	1.57±1.36	4.73±0.60
	2500	0.46±0.13	0.34±0.09	0.05±0.04	1.20±0.30	5.00±0.00	0.00±0.00	2.43±1.99	5.07±0.89
	5000	0.44±0.14	0.34±0.09	0.02±0.03	1.19±0.32	5.00±0.00	0.00±0.00	3.10±1.56	6.20±0.82

Table 2: . Experimentations with fixed-size databases (2000 records) and epochs of varying sizes

Network	Step	KL mean	KL std	KL min	KL max	TP	FN	FP	Time
Alarm	5	9.44±1.36	5.44±0.97	2.32±0.15	19.40±2.67	5.00±0.00	0.00±0.00	1.07±1.12	22.87±2.11
	10	11.81±0.88	7.07±0.54	2.31±0.11	22.40±0.71	5.00±0.00	0.00±0.00	2.03±1.47	25.89±1.96
	15	12.23±1.25	7.53±0.51	2.26±0.12	22.51±0.71	5.00±0.00	0.00±0.00	2.27±1.81	29.99±3.21
	20	12.18±1.16	7.84±0.43	2.34±0.16	22.59±0.53	5.00±0.00	0.00±0.00	2.47±1.67	33.53±2.89
	25	11.70±0.96	7.85±0.42	2.33±0.15	22.43±0.82	5.00±0.00	0.00±0.00	2.40±1.28	38.31±2.91
	30	12.25±1.18	8.14±0.37	2.31±0.16	22.84±0.52	5.00±0.00	0.00±0.00	2.23±1.78	39.88±2.86
Asia	5	0.46±0.11	0.32±0.08	0.05±0.05	1.09±0.31	5.00±0.00	0.00±0.00	1.40±1.50	4.22±0.75
	10	0.48±0.12	0.36±0.09	0.06±0.04	1.26±0.35	5.00±0.00	0.00±0.00	2.27±2.03	4.85±0.70
	15	0.47±0.11	0.37±0.09	0.06±0.05	1.29±0.32	5.00±0.00	0.00±0.00	2.63±1.99	4.66±0.87
	20	0.50±0.11	0.38±0.08	0.07±0.04	1.35±0.31	5.00±0.00	0.00±0.00	2.63±1.66	5.66±0.71
	25	0.49±0.10	0.38±0.09	0.05±0.05	1.34±0.31	5.00±0.00	0.00±0.00	2.73±2.08	5.41±0.79
	30	0.50±0.11	0.39±0.08	0.06±0.05	1.40±0.32	5.00±0.00	0.00±0.00	2.67±2.02	6.00±0.80

(as there are 5 epochs, the best possible value for TP is actually 5). Of course, as "asia" has fewer variables, our algorithm is able to find the true set of transitions with smaller databases (at least 100 records). This clearly makes it appropriate to detect events in dynamic systems. Note however that the number of false positives tends to increase with the size of the database. Actually, when the number of records increases, there are more opportunities to spread the data on low regions of the CPTs and, in addition, the χ^2 test needs data to fit better the CPTs to not reject the null hypothesis (no transition time). However, even in those cases where the algorithm creates erroneous transition times, the network obtained remains very close to the original nsDBN, as highlighted by the Kullback-Leibler distances shown in the first 4 columns. These distances were computed at every time slice and the columns correspond to the average, standard deviation, min and max KL obtained on all the time slices. As the databases are not very large, some random variables' values were never observed in the database at some time slices, hence ruling out Kullback-Leibler comparisons with the original nsDBN. To avoid this problem, we reestimated by maximum likelihood the parameters on the original nsDBNs' structures with the same subdatabases used by our algorithm and we compared these reestimated BNs with those found by our algorithm. In a sense, the reestimated BNs are the "best" BNs we could get provided our databases. Finally, note that the computation times (in seconds) remain quite low and increase sublinearly. This can be explained by the fact that learning new structures is more time consuming than performing χ^2 tests but the latter is performed much more frequently.

In Tables 2 and 3, the number of record was fixed, but we varied the sizes of the epochs (from 5 to 30). It can be noticed that these sizes have no significant effect on the number of true positives or false positives. However, for 500-records databases, the KL distances tend to decrease with the size of the epochs, which makes sense because the algorithm has more opportunities to adjust the CPT's parameters before the next epoch. For 2000-records databases, the KL distances tend to stay level because this database size enabled to estimate very precise CPTs, even for small epochs.

Finally, to conclude these experimentations, we shall have provided comparisons with the algorithm of (**?**) but their framework is very different from ours: they have to process the entire database over all time slices whereas we process it only one slice at a time in streaming. Therefore, they devote a lot of time finding the best transition times when we do not. So a comparison would be unfair, especially on com-

Table 3: . Experimentations with fixed-size databases (500 records) and epochs of varying sizes

Network	Step	KL mean	KL std	KL min	KL max	TP	FN	FP	Time
Alarm	5	14.14±1.24	6.30±0.75	5.03±0.43	24.27±1.54	5.00±0.00	0.00±0.00	0.70±1.04	18.49±1.42
	10	11.96±1.18	6.94±0.69	2.56±1.39	23.03±1.68	5.00±0.00	0.00±0.00	0.97±1.14	15.98±1.19
	15	9.05±1.02	5.77±0.93	1.08±0.67	18.17±2.60	5.00±0.00	0.00±0.00	0.90±0.70	16.43±1.11
	20	7.69±1.22	5.21±1.09	0.76±0.39	16.47±3.77	5.00±0.00	0.00±0.00	1.00±1.00	17.18±1.37
	25	6.95±1.07	4.89±1.09	0.62±0.33	15.69±4.21	5.00±0.00	0.00±0.00	1.53±1.52	17.61±1.34
	30	6.35±0.87	4.22±1.08	0.72±0.55	13.55±4.03	5.00±0.00	0.00±0.00	1.03±0.98	16.43±1.11
Asia	5	0.55±0.11	0.33±0.11	0.14±0.07	1.21±0.40	5.00±0.00	0.00±0.00	0.53±0.88	4.05±0.48
	10	0.57±0.12	0.35±0.11	0.13±0.06	1.27±0.38	5.00±0.00	0.00±0.00	0.90±1.01	4.63±0.56
	15	0.59±0.12	0.36±0.11	0.14±0.06	1.34±0.3715	5.00±0.00	0.00±0.00	1.47±1.12	4.86±0.50
	20	0.60±0.13	0.37±0.12	0.16±0.04	1.37±0.37	5.00±0.00	0.00±0.00	1.17±1.16	4.74±0.56
	25	0.59±0.13	0.36±0.11	0.14±0.06	1.32±0.33	5.00±0.00	0.00±0.00	1.43±1.28	4.91±0.53
	30	0.58±0.13	0.37±0.11	0.12±0.05	1.39±0.40	5.00±0.00	0.00±0.00	2.07±1.44	4.85±0.76

putation times where they exceeded 600 seconds when we are always below 40, and given the very good quality of our learning.

Conclusion

We proposed in this paper a new approach to learn nsDBNs. Our algorithm has very distinct features compared to the other few nsDBN learning algorithms: i) it is specifically designed to be run in streaming mode; ii) it does not make restrictive assumptions on how the structure of the nsDBNs evolves over time; iii) when the structure evolves, it uses a principled approach to enable CPT's parameters dependencies between different epochs. Finally, as shown in the experiments, our algorithm scales is very efficient.