Learning consistent tree-augmented dynamic Bayesian networks *

Margarida Sousa[1,2] and Alexandra M. Carvalho[1,2]

[1] Instituto de Telecomunicações
[2] Instituto Superior Técnico, University of Lisbon, Portugal
{margarida.sousa,alexandra.carvalho}@tecnico.ulisboa.pt

Abstract. Dynamic Bayesian networks (DBNs) offer an approach that allows for causal and temporal dependencies between random variables repeatedly measured over time. For this reason, they have been used in several domains such as medical prognostic predictions, meteorology and econometrics. Learning the intra-slice dependencies is, however, most of the times neglected. This is due to the inherent difficulty in dealing with cyclic dependencies. We propose an algorithm for learning optimal DBNs consistent with the tree-augmented network (tDBN). This algorithm uses the topological order induced by the tDBN to increase its search space exponentially while keeping the time complexity polynomial.

1 Introduction

Bayesian networks (BN) are a powerful probabilistic representation [20] that provide interpretable models of the domain. This is achieved through the definition of a network – a directed acyclic graph (DAG) – that unravels direct conditional dependencies between random variables. This network provides nothing more than a factorization of the joint probability distribution of those variables. Learning a BN from data consists in learning this structure. Having so, it is easy to learn its parameters and make inferences over this probabilistic framework.

Dynamic Bayesian networks (DBN), on the other hand, model stochastic processes [19]. In this case, variables are measured not only once, as for the case of BNs, but repeatedly over time. The networks to be learned consist in a prior network and several transition networks. The prior network is a BN eliciting the dependencies between the random variables at their initial state. The transition network unravels the dynamic dependencies of the variables over time: from past states to current states (inter-slice dependencies); and between current states (intra-slice dependencies).

The inter-slice dependencies are easy to learn as they flow forward in time and do not create cycles [12]. On the other hand, learning the intra-slice dependencies suffers from the hardness of finding an acyclic graph [9,7,11]. A polynomial-time

* This work was supported through FCT, under contract IT (UID/EEA/50008/2013), and by projects PERSEIDS (PTDC/EMS-SIS/0642/2014), NEUROCLINOMICS2 (PTDC/EEI-SII/1937/2014), and internal IT projects QBigData and RAPID.

algorithm for learning optimal DBNs was proposed using the Mutual Information Tests (MIT) [22]. However, learning the inter and intra-slice networks all together is not considered. This step has been done for tree-like networks, resulting in the so-called tree-augmented DBN (tDBN) [17]. We propose to further extend this algorithm by increasing exponentially its search space to networks consistent with the topological order induced by an optimal tDBN. At the same time, we are able to maintain its time complexity polynomial in the size of the input.

The emerging availability of electronic medical records (EMR) is triggering this line of research, bringing large, feature-rich, heterogeneous, noisy, and incomplete time series. The proposed algorithm is currently being used to predict evolution of amyotrophic lateral sclerosis and treatment outcome of arthritis rheumatoid from EMR.

We start by reviewing the basic concepts of both BNs and DBNs. Then, we present the proposed learning algorithm and the experimental results. The paper concludes with a brief discussion and directions for future work.

2 Bayesian networks

Let X denote a *discrete random variable* that takes values over a finite set $\mathcal{X}$ and $\mathbf{X} = (X_1, \ldots, X_n)$ represent an n-dimensional *random vector*, where each X_i takes values in $\mathcal{X}_i = \{x_{i1}, \ldots, x_{ir_i}\}$. Furthermore, let $P(\mathbf{x})$ denotes the probability that $\mathbf{X}$ takes the value $\mathbf{x}$. A *Bayesian network* (BN) encodes the joint probability distribution of a set of n random variables $\{X_1, \ldots, X_n\}$ [20] and it is given by a triple $B = (\mathbf{X}, G, \Theta)$, where:

- $\mathbf{X} = (X_1, \ldots, X_n)$, each random variable X_i taking values in $\{x_{i1}, \ldots, x_{ir_i}\}$, where x_{ik} denotes the k-th value X_i can take.
- $G = (\mathbf{X}, E)$ is a *directed acyclic graph* (DAG) with nodes in $\mathbf{X}$ and edges E representing direct dependencies between the nodes.
- The set Θ encodes the parameters of the network G. Each random variable X_i has an associated *conditional probability distribution* (CPD) a.k.a. local parameters: $\Theta_{ijk} = P_B(X_i = x_{ik} | \Pi_{X_i} = w_{ij})$, where Π_{X_i} denotes the set of parents of X_i in the network G and w_{ij} is the j-th *parent configuration* of Π_{X_i}, which ranges over $\{w_{i1}, \ldots, w_{iq_i}\}$, with $q_i = \prod_{X_j \in \Pi_{X_i}} r_j$.

We note that the random vector $\mathbf{X}$ coincide exactly with the set of nodes in G, and we abuse notation considering that set to be denoted by $\mathbf{X}$.

A BN B induces a unique joint probability distribution over $\mathbf{X}$ given by:

$$P_B(X_1, \ldots, X_n) = \prod_{i=1}^{n} P_B(X_i | \Pi_{X_i}). \tag{1}$$

Intuitively, the graph G of a BN can be viewed as a network structure that provides the skeleton for representing the joint probability, compactly, in a factorized way. This reduces highly the number of parameters needed to describe the full joint probability distribution over the random variables [16,6].

Learning a Bayesian network is done in two steps: first the structure is learned; having the structure fixed, the parameters are learned. This is called *structure learning* and *parameter learning*, respectively. In what follows, we assume data D is complete, i.e, each instance is fully observed, there are no missing values or hidden variables. Moreover, $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ is given by a set of N i.i.d. instances. In that case, N_{ijk} is the number of instances where X_i takes the value x_{ik} and its parents Π_{X_i} takes the configuration w_{ij}. In addition, the number of instances where Π_{X_i} takes the configuration w_{ij} is denoted by N_{ij}.

In order to learn the parameters we assume the underlying graph G is given; in this case, the goal is to estimate the parameters Θ of the network. Using general results of the maximum likelihood estimate we get the following parameters for a BN B:

$$\hat{\theta}_{ijk} = \frac{N_{ijk}}{N_{ij}}, \tag{2}$$

that is denoted by *observed frequency estimates* (OFE).When learning the structure, the aim is to find a DAG G, given D. This can be accomplished through the use of a *scoring function* $\phi : \mathcal{S} \times \mathcal{X} \to \mathbb{R}$, where $\mathcal{S}$ denotes the search space, that measures how well the BN B fits the data D; therefore, it is called *score-based learning* [5,2,3]. The main scoring criteria are Bayesian and information-theoretical [1]. We will focus only on information-theoretical ones, in particular, *log-likelihood* (LL) and *minimum description length* (MDL). The LL of a BN B is given by:

$$\mathrm{LL}(B|D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log(\theta_{ijk}). \tag{3}$$

This criterion does not generalize well as it favors complete network structures, leading to the *overfitting* of the model to the data. The MDL criterion, proposed by Rissanen [21], imposes that the parameters of the model must also be accounted, providing a penalty factor that balances between fitness and model complexity. The MDL is defined by:

$$\mathrm{MDL}(B|D) = LL(B|D) - \frac{1}{2} \ln(N)|B|, \text{ with } |B| = \sum_{i=1}^{n} (r_i - 1) q_i, \tag{4}$$

where $|B|$ corresponds to the number of parameters Θ of the network. These scoring functions have a very important property, they are *decomposable*. This means that the overall score ϕ of B can be expressed as sums of local contributions ϕ_i of each node X_i and its parents (c.f. summations in Eq. (3)). This decomposability property allows for efficient learning procedures based on local-search methods.

In light of the previous discussion, structure learning reduces to an optimization problem: given a scoring function ϕ and a data D, find the BN B that maximizes $\phi(B, D)$.

Learning general BNs is a NP-hard problem [9,7,11]. However, if we restrict the search space $\mathcal{S}$ to branchings (a.k.a. tree-like structures) [8,15] or networks

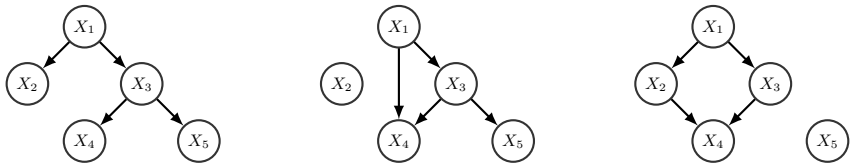


Fig. 1: Given the branching R represented in Figure 1a, Figure 1b represents a C2G w.r.t. R; Figure 1c represents a non-consistent 2-graph w.r.t. R due to the edge from X_2 to X_4.

with bounded in-degree with a known ordering over the variables [10], it is possible to obtain global optimal solutions for this problem. A polynomial-time algorithm for learning BNs with underlying *consistent κ-graphs* (CκG) was proposed combining these ideas [4]. Therein, the authors showed that the set of networks consistent with the optimal branching is exponentially larger, in the number of variables, when comparing with branchings themselves [4]. In addition, the time-complexity of the learning procedure remained polynomial. The method we propose in this paper is an extension of the CκGs to DBNs, so in the following we further introduce notation and detail the CκG learning procedure.

A *κ-graph* is a graph where each node has in-degree at most κ. Given a branching R over a set of nodes V, a graph $G = (V, E)$ is said to be a *consistent κ-graph* (CκG) w.r.t. R if it is a κ-graph and for any edge in E from X_i to X_j the node X_i is in the path from the root of R to X_j. Intuitively, this branching R provides a topological order of the nodes from which the set of parents of each node in the network can be refined without creating cycles, avoiding the hardness of checking for cycles in the DAG. In this way, it is possible to add relevant edges, not considered previously due to the branching restriction (that allows only for one parent), and remove irrelevant ones (as branchings also requires exactly one parent per node, except from the root). For an example see Figure 1.

The algorithm for learning CκG network structures, presented in Algorithm 1, starts by determining an optimal branching R (Step 1); for this it uses the Chow-Liu [8] or Edmond's [13] algorithm (see details in [4]). It then computes the set of candidate ancestors α_i, for each node X_i, compatible with the topological order induced by the optimal branching R (Steps 2–3). The parents of each node X_i in the network are then refined considering those in α_i (Steps 4–9). The algorithm returns a BN of in-degree κ consistent with R, augmenting the search space exponentially, in the number of variables, relatively to branchings, yet keeping a polynomial-time bound in the number of variables n.

3 Dynamic Bayesian networks

Dynamic Bayesian networks (DBN) model the stochastic evolution of a set of random variables over time [19]. Consider the discretization of time in *time slices*

Algorithm 1 Learning CκG networks

```
Run a deterministic algorithm A_φ that outputs an optimal branching R.
for each node X_i in R do
    Compute the set α_i of candidate ancestors for X_i.
    for each subset S of α_i with at most κ nodes do
        Compute φ_i(S, D).
        if φ_i(S, D) is the maximal score for X_i then
            Set Π_{X_i} = S.
        end if
    end for
end for
```

$\mathcal{T} = \{0, \ldots, T\}$. Let $\mathbf{X}[t] = (X_1[t], \ldots, X_n[t])$ be a random vector denoting the value of the set of attributes at time t. Furthermore, let $\mathbf{X}[t_1 : t_2]$ denote the set of random variables $\mathbf{X}$ for the interval $t_1 \leq t \leq t_2$. Consider a set of individuals $\mathcal{H}$ measured over T sequential instants of time. The set of observations is represented as $\{\mathbf{x}^h[t]\}_{h \in \mathcal{H}, t \in \mathcal{T}}$, where $\mathbf{x}^h[t] = (x_1^h, \ldots, x_n^h)$ is a single observation of n attributes, measured at time t and referring to individual h.

In DBNs we aim at defining a probability joint distribution over all possible *trajectories*, i.e., possible values for each attribute X_i and instant t, $X_i[t]$. Let $P(\mathbf{X}[t_1 : t_2])$ denote the joint probability distribution over the trajectory of the process from $\mathbf{X}[t_1]$ to $\mathbf{X}[t_2]$. The space of possible trajectories is enormous, therefore, it is necessary to simplify the problem and make it tractable.

In what follows, observations are viewed as i.i.d. samples of a sequence of probability distributions $\{P_{\theta[t]}\}_{t \in T}$. For all individuals $h \in \mathcal{H}$, and a fixed time t, the probability distribution is considered constant, i.e., $\mathbf{x}^h[t] \sim P_{\theta[t]}, h \in \mathcal{H}$. Using the *chain rule*, the joint probability over $\mathbf{X}$ is given by:

$$P\big(\mathbf{X}[0:T]\big) = P\big(\mathbf{X}[0]\big) \prod_{t=0}^{T-1} P\big(\mathbf{X}[t+1] | \mathbf{X}[0:t]\big).$$

In this case the attributes in time slice $t+1$ depend on all previous time slices t, for $t \in \{0, \ldots, T-1\}$. Usually, not all previous time slices are considered but only a few. In that case, we say that m is the *Markov lag* of the process, also known as m^{th}*-order Markov* process, and so

$$P\big(\mathbf{X}[t+1] | \mathbf{X}[0:t]\big) = P\big(\mathbf{X}[t+1] | \mathbf{X}[t-m+1:t]\big).$$

A further simpification approach is to consider that the process is *stationary*, also called *time invariant* or *homogeneous*, that is, $P\big(\mathbf{X}[t+1] | \mathbf{X}[t]\big)$ is the same for all time slices $t \in \{0, \ldots, T-1\}$. Sometimes, instead of considering the full process as stationary, we consider it *piece-wise stationary*.

In what follows we consider the stochastic process to be a first-order Markov stationary process. This eases the exposition, but its extension to a non-stationary or a m^{th}-order Markov is straightforward. In this case, a *first-order Markov stationary dynamic Bayesian network* (DBN) consists of:

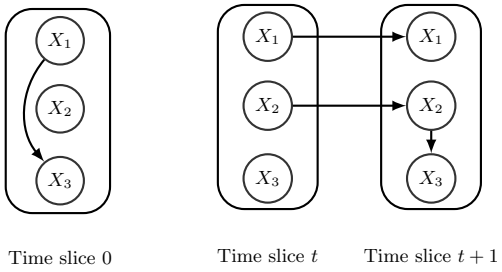


Fig. 2: An example of a DBN. In the left, the prior network B_0 is depicted and in the right, the transition network B_t^{t+1} is represented. The edges $X_1[t] \rightarrow X_1[t+1]$ and $X_2[t] \rightarrow X_2[t+1]$ are the inter-slice connections and edge $X_2[t+1] \rightarrow X_3[t+1]$ represents the intra-slice connection.

- A prior network B^0, which specifies a distribution over the initial states $\mathbf{X}[0]$.
- A transition network B_t^{t+1} over the variables $\mathbf{X}[t:t+1]$, representing the state transition probabilities, for $0 \leq t \leq T-1$.

The transition network has the additional constraint that edges between slices must flow forward in time.

We denote by G_{t+1} the subgraph of B_t^{t+1} with nodes $\mathbf{X}[t+1]$ that contains only the intra-slice dependencies. Observe that a transition network encodes the *inter-slice dependencies*, from time transitions t to $t+1$, and *intra-slice dependencies*, in time slice $t+1$ only. Figure 2 depicts an example of a DBN.

Learning dynamic Bayesian networks, considering no hidden variables or missing values, i.e., considering a fully observable process, reduces simply to learning two BNs: the initial network B_0 and the transition network B_t^{t+1}, taking into account that in B_t^{t+1} edges between slices must flow forward in time [14]. Not considering the acyclicity constraints, it was proved that learning a BN does not have to be NP-hard [12]. This result can be applied to DBNs, as the resulting *unrolled graph*, that contains a copy of each attribute in each time step, is acyclic. For this reason, several methods that consider only inter-slice dependencies appeared, as therein no cycles can arise [22,18].

More recently, a polynomial-time algorithm was proposed that learns both the inter and intra-slice connections in a transition network; the resultant network was denoted by tree-augmented DBN (tDBN) [17]. Therein, the search space for the intra-slice networks was restricted to have a tree-like structure; each attribute in time slice $t+1$ was allowed to have at most one parent from the same time slice, and up to p parents were allowed from previous time slices; p is a user-input parameter.

We now describe the first-order Markov stationary tDBN algorithm. Let $\mathcal{P}_{\leq p}(\mathbf{X}[t])$ be the set of subsets of $\mathbf{X}[t]$ with cardinality less or equal to p. For each $X_i[t+1] \in \mathbf{X}[t+1]$, the optimal set of parents $\Pi_{X_i[t+1]} \in \mathcal{P}_{\leq p}(\mathbf{X}[t])$ yields

the following score:

$$s_i = \max_{\Pi_{X_i[t+1]} \in \mathcal{P}_{\leq p}(\mathbf{X}[t])} \phi_i(\Pi_{X_i[t+1]}, D_t^{t+1}),$$

where ϕ_i is the local score of attribute $X_i[t+1]$ and D_t^{t+1} is the subset of observations for time transition $t \to t+1$. Then, allowing at most one parent $X_j[t+1]$ from the current time slice, the maximal score is defined as:

$$s_{ij} = \max_{\Pi_{X_i[t+1]} \in \mathcal{P}_{\leq p}(\mathbf{X}[t])} \phi_i(\Pi_{X_i[t+1]} \cup \{X_j[t+1]\}, D_t^{t+1}). \tag{5}$$

A complete directed graph is built such that each edge $X_j[t+1] \to X_i[t+1]$ has the following weight,

$$e_{ij} = s_{ij} - s_i, \tag{6}$$

that is, the gain in the network score of adding $X_j[t+1]$ as a parent of $X_i[t+1]$. Herein, the tDBN algorithm is able to determine the optimal set of inter and intra-slice parents of $X_i[t+1]$ in a one-step procedure.

Generally $e_{ij} \neq e_{ji}$, as the edge $X_i[t+1] \to X_j[t+1]$ may account for the contribution from the inter-slice parents and, in general, inter-slice parents of $X_i[t+1]$ and $X_j[t+1]$ are not the same. Therefore, Edmond's algorithm [13] is applied to obtain a maximum branching for the intra-slice network.

The pseudo-code of the procedure is given in Algorithm 2. A complete directed graph in $\mathbf{X}[t+1]$ is built (Step 1). Afterwards, in Step 2, the weight of all edges and the optimal set of parents for all nodes are determined according to Eq. (6) for a given scoring criterion ϕ. An optimal branching is obtained using Edmonds' algorithm [13] in Step 3. Step 4 retrieves the tree-like intra-slice transition network elicited in Step 3 with the optimal inter-slice parents determined in Step 2.

Algorithm 2 Optimal first-order Markov stationary tDBN

1: Build a complete directed graph in $\mathbf{X}[t+1]$.
2: Calculate the weight of all edges and the optimal set of parents of all nodes.
3: Apply Edmonds' algorithm to retrieve an optimal branching.
4: Extract transition network $t \to t+1$.

The tDBN algorithm has a worst-case time complexity that is linear in N (size of the input data), polynomial in n (number of variables) and r (number of values a variable can take), and exponential in p (number of parents from the previous time slice).

4 Proposed method

Profiting from the CκG learning algorithm for BN, we propose an algorithm to learn DBN structures consistent with the tDBN. In what follows, as for tDBN,

the proposed method is explained only for first-order Markov stationary DBNs; the extension to non-stationary m^{th}-order Markov, however, is straightforward.

Rigorously, a DBN is said to be a CκG, denoted by cDBN, if the intra-slice transition network G_{t+1} is a κ-graph where each edge from $X_i[t+1]$ to $X_j[t+1]$ is consistent with the intra-slice tree-network of a given tDBN. Moreover, each node $X_i[t+1]$ has at most p parents from the previous time slice. Therefore, in order to be well-defined, a cDBN needs two positive integers: κ and p. In addition, the given tDBN is an optimal tDBN computed with exactly the same number of p parents from the previous time slice.

We now describe briefly the proposed algorithm. It starts by computing an optimal tDBN. The intra-slice branching G_{t+1} is then used to refine the set of parents of each node in the network at time-slice $t+1$ so that they are consistent with the topological order induced by such branching. This is done by computing the candidate ancestors of each node $X_i[t+1]$, denoted by $\alpha_{i,t+1}$; these are exactly the set of nodes in $t+1$ connecting the root of the optimal branching given by G_{t+1} and $X_i[t+1]$. For node $X_i[t+1]$, the optimal set of past parents $\mathbf{X}_{ps}[t]$ and intra-slice parents, denoted by $\mathbf{X}_{ps}[t+1]$, are obtained in a one-step procedure by finding

$$\max_{\mathbf{X}_{ps}[t]\in\mathcal{P}_{\leq p}(\mathbf{X}[t])} \max_{\mathbf{X}_{ps}[t+1]\in\mathcal{P}_{\leq\kappa}(\alpha_{i,t+1})} \phi_i(\mathbf{X}_{ps}[t]\cup\mathbf{X}_{ps}[t+1], D_t^{t+1}), \tag{7}$$

where $\mathcal{P}_{\leq\kappa}(\alpha_{i,t+1})$ is the set of all subsets of $\alpha_{i,t+1}$ of cardinality less than or equal to κ. Note that, if $X_i[t+1]$ is the root, $\mathcal{P}_{\leq\kappa}(\alpha_{i,t+1}) = \{\emptyset\}$, so the set of intra-slice parents $\mathbf{X}_{ps}[t+1]$ of $X_i[t+1]$ is always empty.

Algorithm 3 finds an optimal first-order Markov stationary cDBN, given a decomposable scoring criterion ϕ, a set of n random variables, a maximum number of parents from the previous time slice of p, and a bounded in-degree in the intra-slice network of κ.

Algorithm 3 Learning optimal first-order Markov stationary cDBN

```
Compute an optimal tDBN with p parents with intra-slice graph given by G_{t+1}.
for each node X_i[t+1] ∈ G_{t+1} do
    Compute the set α_{i,t+1} of ancestors of X_i[t+1].
    for each subset P in P_{≤p}(X[t]) do
        for each subset S in P_{≤κ}(α_{i,t+1}) do
            Compute φ_i(P ∪ S, D_t^{t+1}).
            if φ_i(P ∪ S, D_t^{t+1}) is the maximal score for X_i[t+1] then
                Set Π_{X_i[t+1]} = P ∪ S.
            end if
        end for
    end for
end for
```

The proposed algorithm increases exponentially the search space of the intra-slice transition network. Indeed, in the context of BNs, it was proved that the

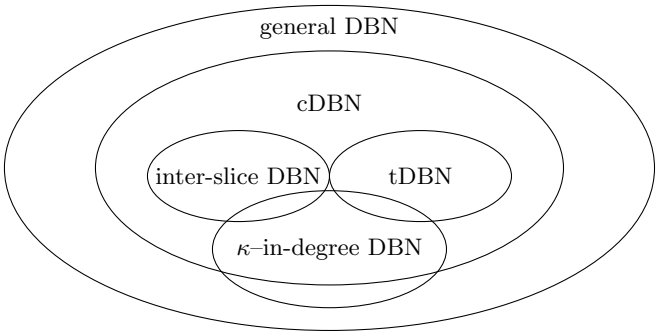


Fig. 3: Search-space classes of first-order Markov DBNs discussed in this paper. The class of inter-slice DBN contains all DBNs with no intra-slice dependencies. The class tDBN contains tree-augmented DBNs for all p parents from the previous time slice. The cDBN class contains all $(\kappa + p)$–in-degree cDBNs for all p and κ. The class of κ–in-degree DBN contains DBNs with in-degree at most $\kappa < 2n$, where n is the number of variables per time slice. This class does not include the tDBN as κ may be smaller than p. The general DBN class coincides with the $(2n - 1)$–in-degree DBNs.

class of CκGs is exponentially larger, in the number of variables, when compared to tree-network structures [4], result which is straightforwardly extended to cDBNs. In Figure 3 the search-space classes relating DBNs, namely tDBNs and cDBNs, are presented.

In terms of worst-time complexity, when comparing with the tDBN algorithm, Algorithm 3 is linear in N (size of the input data) and T (number of time slices), polynomial in n (number of variables) and r (number of values a variable can take), and exponential in p (number of parents from the previous time slice t) and κ (number of parents in current time slice $t + 1$).

5 Experimental results

We evaluate the proposed algorithm comparing it with the tDBN learning algorithm [17]. Our algorithm was implemented in Java and was released under a free software license.[3] The experiments were run on an Intel Core i5-3320M CPU @ 2.60GHz×4 machine.

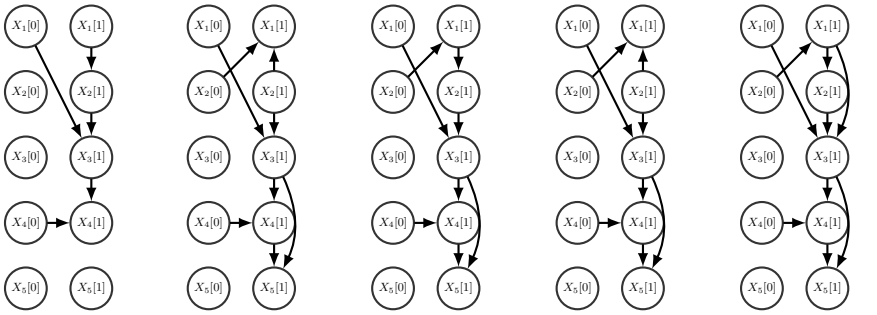
We analyze the performance of the proposed algorithm for synthetic data generated from first-order Markov stationary cDBNs. Four cDBN structures and parameters were determined, and observations were sampled from the generated networks, for a given number of observations N. The parameters p and κ were taken to be the maximum in-degree of the inter and intra-slice network, respectively, of the transition network considered. The four transition networks

[3] `https://margaridanarsousa.github.io/learn_cDBN/`

considered included: (i) one incomplete cDBN with $n = 5$, $\kappa = 2$ and at most $p = 1$ parents from the previous time slice; (ii) one complete cDBN with $n = 5$, $\kappa = 4$ and at most $p = 1$ parents from the previous time slice; (iii) one incomplete cDBN with with $n = 10$, $\kappa = 6$ and at most $p = 1$ parents from the previous time slice; (iv) one incomplete cDBN with $n = 10$, $\kappa = 4$ and at most $p = 1$ parents from the previous time slice. The tDBN and cDBN algorithms were applied to the resultant data sets, and the ability to learn and recover the original network structure was measured using the precision, recall and F_1-measure metrics.Two scoring functions were used: LL in Eq. (3) and MDL in Eq. (4).

The results are depicted in Table 1 and the presented values are annotated with a 95% confidence interval, over 5 trials. Considering LL, the cDBN algorithm consistently outperforms tDBN, for all number of instances N considered. As for MDL, the cDBN networks have a greater number of parameters, therefore the model complexity penalization factor of MDL leads to the selection of simple networks when considering a low number of instances. Hence, in these cases, the tDBN+MDL gives raise to better results. Generally, considering $N \geq 1000$ instances for the networks considered, cDBN+MDL outperforms tDBN+MDL. Comparing the results for networks 1 and 2, we observe that LL gives raise to better results when considering complete networks, whereas considering less complex structures, the MDL has better results. On the other hand, when comparing the results for networks 2:4 and 3:4, we conclude that considering a higher number of nodes and intra-slice in-degree κ, respectively, a higher number of instances is necessary to achieve similar recalls.

In Figure 4, an example of the cDBN+MDL learning algorithms ability to recover a known network is shown. The original cDBN network has $n = 5$ attributes, each taking $r = 2$ different values, having up to one parent from the previous time slice and two from the current time slice. Varying the number of input observations N, five recovered networks are shown. As N increases, the recovered network structures become more similar to the original, converging to the original for $N = 1800$.



(a) $N = 100$. (b) $N = 500$. (c) $N = 1000$. (d) $N = 1500$. (e) $N = 1800$.

Fig. 4: Reconstructed networks for cDBN algorithm, where N is the number of instances used to learn. The true network was recovered when $N = 1800$.

N	tDBN+LL				tDBN+MDL				cDBN+LL				cDBN+MDL			
	Pre	Rec	F_1	Time	Pre	Rec	F_1	Time	Pre	Rec	F_1	Time	Pre	Rec	F_1	Time
Network 1 ($n = 5, p = 1, \kappa = 2, r = 3$)																
100	60 ± 5	60 ± 5	60 ± 5	0	92 ± 14	51 ± 8	66 ± 10	0	58 ± 5	76 ± 7	65 ± 6	0	100 ± 0	20 ± 4	33 ± 6	0
500	78 ± 0	78 ± 0	78 ± 0	0	86 ± 8	64 ± 4	74 ± 5	0	73 ± 3	98 ± 4	84 ± 3	0	98 ± 4	84 ± 5	90 ± 4	0
1000	78 ± 0	78 ± 0	78 ± 0	0	88 ± 0	78 ± 0	82 ± 0	0	75 ± 0	100 ± 0	86 ± 0	0	100 ± 0	100 ± 0	100 ± 0	0
2000	78 ± 0	78 ± 0	78 ± 0	0	88 ± 0	78 ± 0	82 ± 0	0	75 ± 0	100 ± 0	86 ± 0	0	100 ± 0	100 ± 0	100 ± 0	0
Network 2 ($n = 5, p = 1, \kappa = 4, r = 3$)																
100	71 ± 10	43 ± 6	53 ± 7	0	62 ± 13	19 ± 6	29 ± 8	0	71 ± 3	56 ± 3	63 ± 3	0	0 ± 0	4 ± 3	0 ± 0	0
500	96 ± 5	57 ± 3	72 ± 4	0	96 ± 7	41 ± 7	58 ± 8	0	98 ± 3	77 ± 3	87 ± 3	0	90 ± 18	28 ± 9	42 ± 13	0
1000	98 ± 4	59 ± 2	73 ± 3	0	100 ± 0	47 ± 0	64 ± 0	0	100 ± 0	80 ± 0	89 ± 0	0	100 ± 0	44 ± 3	61 ± 3	0
2000	100 ± 0	60 ± 0	75 ± 0	0	100 ± 0	52 ± 2	68 ± 2	0	100 ± 0	80 ± 0	89 ± 0	0	100 ± 0	64 ± 5	78 ± 3	0
Network 3 ($n = 10, p = 1, \kappa = 6, r = 3$)																
100	53 ± 5	33 ± 3	41 ± 4	0	66 ± 8	23 ± 4	34 ± 5	0	36 ± 9	38 ± 7	37 ± 8	2	83 ± 18	7 ± 2	13 ± 4	4
500	72 ± 5	45 ± 3	56 ± 4	0	88 ± 5	40 ± 2	55 ± 3	0	53 ± 2	68 ± 7	60 ± 4	1	100 ± 0	33 ± 2	50 ± 2	1
1000	77 ± 2	49 ± 1	60 ± 2	0	92 ± 2	46 ± 1	61 ± 2	0	59 ± 2	75 ± 4	66 ± 2	2	100 ± 0	47 ± 0	64 ± 0	7
2000	78 ± 2	49 ± 1	60 ± 1	0	92 ± 2	48 ± 1	63 ± 2	0	60 ± 1	78 ± 2	68 ± 2	10	100 ± 0	58 ± 3	73 ± 2	8
Network 4 ($n = 10, p = 1, \kappa = 4, r = 3$)																
100	29 ± 9	23 ± 7	26 ± 8	0	36 ± 17	13 ± 6	19 ± 9	0	24 ± 5	33 ± 7	28 ± 6	0	40 ± 33	3 ± 2	0 ± 0	0
500	58 ± 3	46 ± 2	51 ± 3	0	80 ± 10	33 ± 4	47 ± 6	0	43 ± 7	61 ± 12	50 ± 8	0	73 ± 14	31 ± 8	43 ± 10	3
1000	60 ± 5	48 ± 4	53 ± 4	0	80 ± 8	38 ± 3	51 ± 5	0	41 ± 6	69 ± 9	51 ± 7	4	86 ± 6	48 ± 4	62 ± 5	24
2000	65 ± 2	52 ± 2	58 ± 2	0	86 ± 9	48 ± 4	62 ± 5	0	50 ± 3	74 ± 7	59 ± 1	17	85 ± 10	68 ± 9	76 ± 9	17

Table 1: Comparative structure recovery results for tDBN and cDBN on simulated data. The tDBN+LL and tDBN+MDL denote, respectively, the tDBN learning algorithm with LL and MDL criteria. Similarly, for cDBN+LL and cDBN+MDL. For each network, n is the number of variables, p is the maximum inter-slice in-degree, κ is the maximum intra-slice in-degree, and r is the number of values of all attributes. On the left, N is the number of observations. Precision (Pre), recall (Rec) and F_1-measure (F_1) values are presented as percentages, running time is in seconds.

6 Conclusions

We conclude that the proposed algorithm allows to learn efficiently DBNs consistent with the topological order induced by the transition network of an optimal tDBN as far as the in-degree bounds p and κ are kept low. Notwithstanding, it is well known that in most practical scenarios BNs behave well with small in-degree network structures.

The resulting method is scalable (in the number of instances N, number of time slices T and number of variables n) and therefore suitable for the increasing amount of temporal data arising from medicine (and also other fields). We are currently using cDBN to predict the class of evolution of Amyotrophic Lateral Sclerosis (ALS) patients and the treatment outcome of rheumatoid arthritis (RA). These ALS and RA data is collected as a multivariate time series with heterogeneous values, which can be addressed effectively by cDBN. DBNs play the unique role of not only being able to model evolution in time of several autocorrelated variables but also provide models that are human interpretable.

Further improvements of the algorithm may include using a total order, instead of a partial one (as the topological order), and extend the learning procedure to allow hidden variables.

References

1. A. M. Carvalho. Scoring functions for learning Bayesian networks. *INESC-ID Tec. Rep*, 12, 2009.
2. A. M. Carvalho, P. Adão, and P. Mateus. Efficient approximation of the conditional relative entropy with applications to discriminative learning of Bayesian network classifiers. *Entropy*, 15(7):2716–2735, 2013.
3. A. M. Carvalho, P. Adão, and P. Mateus. Hybrid learning of Bayesian multinets for binary classification. *Pattern Recognition*, 47(10):3438–3450, 2014.
4. A. M. Carvalho and A. L. Oliveira. Learning Bayesian networks consistent with the optimal branching. In *Proc. ICMLA' 07*, pages 369–374. IEEE, 2007.
5. A. M. Carvalho, T. Roos, A. L. Oliveira, and P. Myllymäki. Discriminative learning of Bayesian networks via factorized conditional log-likelihood. *Journal of Machine Learning Research*, 12:2181–2210, 2011.
6. E. Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50–63, 1991.
7. D. M. Chickering. Learning Bayesian networks is NP-complete. *Learning from Data: Artificial Intelligence and Statistics V*, 112:121–130, 1996.
8. C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.
9. G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.
10. G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.
11. P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993.
12. N. Dojer. Learning Bayesian networks does not have to be NP-hard. In *Proc. MFCS'06*, pages 305–314. Springer, 2006.
13. J. Edmonds. Optimum branchings. *Mathematics and the Decision Sciences, Part*, 1:335–345, 1968.
14. N. Friedman, K. Murphy, and S. Russell. Learning the structure of dynamic probabilistic networks. In *Proc. UAI'98*, pages 139–147, 1998.
15. D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.
16. D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
17. J. L. Monteiro, S. Vinga, and A. M. Carvalho. Polynomial-time algorithm for learning optimal tree-augmented dynamic Bayesian networks. In *Proc. UAI'15*, pages 622–631, 2015.
18. K. P. Murphy. The Bayes Net Toolbox for MATLAB. *Computing Science and Statistics*, 33:2001, 2001.
19. K. P. Murphy and S. Russell. Dynamic Bayesian networks: representation, inference and learning. 2002.
20. J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.
21. J. Rissanen. *Minimum description length principle*. Wiley Online Library, 1985.
22. N. X. Vinh, M. Chetty, R. Coppel, and P. P. Wangikar. Polynomial time algorithm for learning globally optimal dynamic Bayesian network. In *Proc. ICONIP'11*, pages 719–729. Springer, 2011.