Adapting Bayes Network Structures to Non-stationary Domains

Søren Holbech Nielsen
Department of Computer Science
Aalborg University, Denmark

Thomas D. Nielsen
Department of Computer Science
Aalborg University, Denmark

Abstract

When an incremental structural learning method gradually modifies a Bayesian network (BN) structure to fit a sequential stream of observations, we call the process structural adaptation. Structural adaptation is useful when the learner is set to work in an unknown environment, where a BN is gradually being constructed as observations of the environment are made. Existing algorithms for incremental learning assume that the samples in the database have been drawn from a single underlying distribution. In this paper we relax this assumption, so that the underlying distribution can change during the sampling of the database. The proposed method can thus be used in unknown environments, where it is not even known whether the dynamics of the environment are stable. We state formal correctness results for our method, and demonstrate its feasibility experimentally.

Keywords: Bayesian networks, learning, adaptation, non-stationary domains.

1 Introduction

Ever since Pearl (1988) published his seminal book on Bayesian networks (BNs), the formalism has become a widespread tool for representing, eliciting, and discovering probabilistic relationships. One area of research that has seen much activity is the area of structural learning of BNs. Here probabilistic relationships for variables are discovered, or inferred, from a database of observations of these variables (see e.g. (Heckerman, Geiger, and Chickering 1995)). One part of this research area focuses on incremental structural learning, where observations are received sequentially, and a BN structure is gradually constructed along the way without keeping all observations in memory. A special case of incremental structural learning is structural adaptation, where the incremental algorithm maintains one or more candidate structures and applies changes to these structures as observations are received. This particular area of research has received little attention, with the only results that we are aware of being (Buntine 1991; Lam and Bacchus 1994; Lam 1998; Friedman and Goldszmidt 1997; Roure 2004).

A common characteristic of these results is that they all assume that the database of observations has been produced by a stationary stochastic process. That is, the ordering of the observations in the database is inconsequential. However, many real life observable processes cannot really be said to be invariant with respect to time: Mechanical mechanisms may suddenly fail, for instance, and non-observable effects may change abruptly. When human decision makers are somehow involved in the data generating process, they are almost surely not fully describable by the observables and may change their behavior instantaneously. A simple example of a situation in which it is unrealistic to expect a stationary generating process is an industrial system, in which some component is exchanged for one of another make. Similarly, if the coach of a soccer team changes the strategy of the team during a match, data on the play from after the change would be distributed differently from the data representing the time before.

In this paper we relax the assumption on stationary data, opting instead at learning from data which is only "approximately" stationary. More concretely, we assume that the data generating process is piecewise stationary, as in the examples given above. Thus, we do not try to deal with situations in which the data generating process changes gradually, as can happen when machinery is slowly being worn down.[1] Furthermore, we focus on domains in which the shifts in distribution from one stationary period to the next is of a local nature (i.e. only a subset of the probabilistic relationships among variables change as a shift takes place).

2 Preliminaries

To start off we present the definitions and terminology used in the remainder of the text. As a general notational rule we use bold font to denote sets and vectors ($\boldsymbol{V}$, $\boldsymbol{c}$, etc.) and calligraphic font to denote mathematical structures and compositions ($\mathcal{B}$, $\mathcal{G}$, etc.). Moreover, we shall use upper case letters to denote random variables or sets of random variables (X, Y, $\boldsymbol{V}$, etc.), and lower case letters to denote specific states of these variables (x_4, y', $\boldsymbol{c}$, etc.).

A BN $\mathcal{B} \equiv (\mathcal{G}, \boldsymbol{\Phi})$ over a set of discrete variables $\boldsymbol{V}$ consists of an acyclic directed graph (traditionally abbreviated DAG) $\mathcal{G}$, whose nodes are the variables in $\boldsymbol{V}$, and a set of conditional probability distributions $\boldsymbol{\Phi}$ (which we abbreviate CPTs for "conditional probability table"). A unique joint distribution $P_{\mathcal{B}}$ over $\boldsymbol{V}$ is obtained by taking the product of all the CPTs in $\boldsymbol{\Phi}$.

For any graph $\mathcal{G} \equiv (\boldsymbol{V}, \boldsymbol{E} \subseteq \boldsymbol{V} \times \boldsymbol{V})$ we shall use $X \rightarrow Y$ to denote that there is an arc from X to Y in $\mathcal{G}$, i.e. the fact that $(X, Y) \in \boldsymbol{E}$ and $(Y, X) \notin \boldsymbol{E}$, and $X - Y$ to denote that there is a link between X and Y, $\{(X, Y), (Y, X)\} \subseteq \boldsymbol{E}$. In addition to $\mathrm{pa}_{\mathcal{G}}(X)$, we introduce the notation $\mathrm{ch}_{\mathcal{G}}(X)$, $\mathrm{adj}_{\mathcal{G}}(X)$, and $\mathrm{de}_{\mathcal{G}}(X)$ to mean the children, adjacents, and descendants of node X in $\mathcal{G}$, respectively. When $\mathcal{G}$ is obvious from the context we shall leave out the subscript.

Due to the construction of $P_{\mathcal{B}}$ we are guaranteed that all dependencies inherent in $P_{\mathcal{B}}$ can be read directly from $\mathcal{G}$ using the *d-separation criterion* (Pearl 1988). The d-separation criterion states that, if X and Y are d-separated by $\boldsymbol{Z}$, then it holds that X is conditionally independent of Y given $\boldsymbol{Z}$ in $P_{\mathcal{B}}$, or equivalently, if X is conditionally dependent of Y given $\boldsymbol{Z}$ in $P_{\mathcal{B}}$, then X and Y are not d-separated by $\boldsymbol{Z}$ in $\mathcal{G}$. In the remainder of the text, we use $X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{Z}$ to denote that X is d-separated from Y by $\boldsymbol{Z}$ in the DAG $\mathcal{G}$, and $X \perp\!\!\!\perp_{P} Y \mid \boldsymbol{Z}$ to denote that X is conditionally independent of Y given $\boldsymbol{Z}$ in the distribution P. The d-separation criterion is thus

$$X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{Z} \Rightarrow X \perp\!\!\!\perp_{P_{\mathcal{B}}} Y \mid \boldsymbol{Z}, \tag{1}$$

for any BN $\mathcal{B} \equiv (\mathcal{G}, \boldsymbol{\Phi})$. The set of all conditional independence statements that may be read from a graph in this manner is referred to as that graph's *d-separation properties*. We refer to any two graphs over the same variables as being *equivalent* if they have the same d-separation properties. Equivalence is obviously an equivalence relation.

For a DAG $\mathcal{G}$, we define the *pattern* of $\mathcal{G}$ as the graph $\mathcal{G}^*$ obtained from the skeleton of $\mathcal{G}$ by directing links that participate in a v-structure[2] in $\mathcal{G}$ in the direction dictated by $\mathcal{G}$. Verma and Pearl (1991) proved:

Theorem 1. *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be DAGs over $\boldsymbol{V}$. $\mathcal{G}_1$ is equivalent to $\mathcal{G}_2$ iff $\mathcal{G}_1^* = \mathcal{G}_2^*$.*

This equivalence relation defines an equivalence class, and for any member G the class is uniquely represented by the pattern $\mathcal{G}^*$. Any graph $\mathcal{G}'$ obtained from $\mathcal{G}^*$ by directing the

[1] The changes in distribution of such data is of a continuous nature, and adaptation of networks would probably be better accomplished by adjusting parameters in the net, rather than the structure itself.

[2] Two arcs $X \rightarrow Y$ and $Z \rightarrow Y$ in $\mathcal{G}$ constitute a v-structure if X and Z are non-adjacent in $\mathcal{G}$.

remaining undirected links, without creating a directed cycle or a new v-structure, is then equivalent to $\mathcal{G}$. We say that $\mathcal{G}'$ is a *consistent extension* of $\mathcal{G}^*$. The partially directed graph $\mathcal{G}^{**}$ obtained from $\mathcal{G}^*$, by directing undirected links as they appear in $\mathcal{G}$ whenever all consistent extensions of $\mathcal{G}^*$ agree on this direction, is called the *completed* pattern of $\mathcal{G}$. $\mathcal{G}^{**}$ is obviously a unique representation of $\mathcal{G}$'s equivalence class as well. Any arc in $\mathcal{G}^{**}$ is called *compelled* in $\mathcal{G}$.

Given any joint distribution P over $\boldsymbol{V}$ it is possible to construct a BN $\mathcal{B}$ such that $P = P_{\mathcal{B}}$ (Pearl 1988). A distribution P for which there is a BN $\mathcal{B}_P \equiv (\mathcal{G}_P, \boldsymbol{\Phi}_P)$ such that $P_{\mathcal{B}_P} = P$ and for which

$$X \perp\!\!\!\perp_P Y \mid \boldsymbol{Z} \Rightarrow X \perp\!\!\!\perp_{\mathcal{G}_P} Y \mid \boldsymbol{Z} \tag{2}$$

holds, is called *DAG faithful*, and $\mathcal{B}_P$ (and sometimes $\mathcal{G}_P$ alone) is called a *perfect map*. DAG faithful distributions are important, since if a data generating process is known to be DAG faithful, then a perfect map can, in principle, be inferred from the data under the assumption that the data is representative of the distribution.

For any probability distribution P over variables $\boldsymbol{V}$ and variable $X \in \boldsymbol{V}$, we define a *Markov boundary* of X (denoted by $\mathrm{mb}_P(X)$) to be a set $\boldsymbol{Z} \subseteq \boldsymbol{V} \setminus \{X\}$ such that $X \perp\!\!\!\perp_P \boldsymbol{V} \setminus (\boldsymbol{Z} \cup \{X\}) \mid \boldsymbol{Z}$ and this holds for no proper subset of $\boldsymbol{Z}$. Pearl (1988) proved that if G is a perfect map of P over $\boldsymbol{V}$ and $X \in \boldsymbol{V}$, then the Markov boundary of X is unique and consists of X's parents, children, and children's parents in G (denoted by $\mathrm{mb}_{\mathcal{G}}(X)$).

3 The Adaptation Problem

We will work with sequences of observations that are samples from a *piecewise DAG faithful distribution*, meaning that the sequence can be partitioned into sets such that each set is a database sampled from a single DAG faithful distribution. Formally, let $\mathcal{D} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_l)$ be a data stream over variables $\boldsymbol{V}$. We say that $\mathcal{D}$ is *sampled from a piecewise DAG faithful distribution* (or simply that it is a piecewise DAG faithful sequence) if there are indices $1 = i_1 < \cdots < i_{m+1} = l + 1$, such that each $\mathcal{D}_j \equiv (\boldsymbol{v}_{i_j}, \ldots, \boldsymbol{v}_{i_{j+1}-1})$, for $1 \leq j \leq m$, is a sequence of samples from a single DAG faithful distribution. The *rank* of the sequence is the size of the smallest such partition, i.e. $\min_j i_{j+1} - i_j$, and we say that m is its *size* and l its *length*. A pair of consecutive samples, $\boldsymbol{v}_i$ and $\boldsymbol{v}_{i+1}$, constitute a *shift* in $\mathcal{D}$, if there is j such that $\boldsymbol{v}_i \in \mathcal{D}_j$ and $\boldsymbol{v}_{i+1} \in \mathcal{D}_{j+1}$. Obviously, by selecting the partitions small enough we can have any sequence of observations being indistinguishable from a piecewise DAG faithful sequence, so we restrict our attention to sequences that are piecewise DAG faithful of at least rank r. However, we do not assume that neither the actual rank nor size of the sequences are known, and specifically we do not assume that the indices $i_1, \ldots, i_{m+1}$ are known.

The learning task that we address consists of incrementally learning a BN, while receiving a piecewise DAG faithful sequence of samples, and making sure that after each sample point the BN structure is as close as possible to the distribution that generated this point. Throughout the paper we assume that each sample is complete, so that no observations in the sequence have missing values. Formally, let $\mathcal{D}$ be a complete piecewise DAG faithful sample sequence of length l, and let P_i be the distribution generating sample point $\boldsymbol{v}_i$. Furthermore, let $\mathcal{B}_1, \ldots, \mathcal{B}_l$ be the BNs found by a structural adaptation method M when receiving $\mathcal{D}$. Given a distance measure *dist* on BNs, we define the *deviance* of M on $\mathcal{D}$ wrt *dist* as

$$dev(M, \mathcal{D}) \equiv \frac{1}{l} \sum_{i=1}^{l} dist(\mathcal{B}_{P_i}, \mathcal{B}_i) \ .$$

We say that a method M *adapts* to $\mathcal{D}$ wrt *dist* if M seeks to minimize its deviance on $\mathcal{D}$ wrt *dist*.

That a method aggressively adapts to a piecewise DAG faithful sample sequence might come at a price: Every time the method learns a new BN different from the previous one, the user of the learned BNs would have to inspect the new network and possibly replan accordingly. Similarly, the computational resources used for learning a new network might be better used for other purposes, if the newly learned BN is only marginally closer to representing the generating distribution than the currently held one. Therefore, we introduce a measure capturing the average improvement achieved by each new learned network: Given the distance measure *dist* on BNs, we define the *efficiency* of M on $\mathcal{D}$ wrt *dist* as

$$\mathit{eff}(M, \mathcal{D}) \equiv \frac{1}{|\{i \,:\, \mathcal{B}_i \neq \mathcal{B}_{i-1}\}|} \sum_{i \,:\, \mathcal{B}_i \neq \mathcal{B}_{i-1}} \left(\mathit{dist}(\mathcal{B}_{P_{i-1}}, \mathcal{B}_{i-1}) - \mathit{dist}(\mathcal{B}_{P_i}, \mathcal{B}_i) \right) .$$

An efficiency close to zero would then mean that the method improves on the average — but not much. A higher number would mean that the method improves more drastically when it changes the net. A negative efficiency is an indication that on the average the method does more wrong than good. Note, however, that efficiency in itself cannot be used to judge an adaptation method, as the measure is nearly independent of how well the learned networks actually fit the underlying distributions. For instance, a learning method that learns only once, at the reception of the last observation (where the generating distribution has changed much) would have a good chance of scoring a high efficiency, but clearly it is not a good method for adaptation. Conversely, the perfect learner, which always output the correct network, would achieve a neutral score of 0. However, if two methods tend to yield comparable deviances, efficiency becomes a relevant measure.[3]

4 A Structural Adaptation Method

The method proposed here continuously monitors the data stream $\mathcal{D}$ and evaluates whether the last, say k, observations fit the current model. When this turns out not to be the case, we conclude that a shift in $\mathcal{D}$ took place k observations ago. To adapt to the change, an immediate approach could be to learn a new network from the last k cases. By following this approach, however, we will unfortunately loose all the knowledge gained from cases before the last k observations. This is a problem if some parts of the perfect maps of the two distributions on each side of the shift do not differ, since in such situations we relearn those parts from the new data, even though they have not changed. Not only is this a waste of computational effort, but it can also be the case that the last k observations, while not directly contradicting these parts, do not enforce them either, and consequently they are altered erroneously. Instead, we try to detect where changes have taken place in the perfect maps of the two distributions, and only learn these parts. This presents challenges not only in detection, but also in learning the changed parts and combining them with the non-changed parts. Hence, the method consists of two main mechanisms: One, monitoring the current BN while receiving observations and detecting when and where the model should be changed, and two, relearning the parts of the model that conflicts with the observations, and integrating the relearned parts with the remaining parts of the model. These two mechanisms are described below in Sections 4.1 and 4.2, respectively.

[3] An anonymous reviewer has suggested to us that it may be interesting to split the efficiency measure into two measures, representing the positive and negative contributions to the measure, and analyze these in isolation. In particular, this could be informative in situations where e.g. a single negative term is dominating the measure. Sadly, we have not had time to pursue this line of analysis in this paper.

4.1 Detecting Changes

The detection part of our method, outlined in Algorithm 1, continuously processes the cases it receives. For each observation $\boldsymbol{v}$ and node X, the method measures (using ConflictMeasure($\mathcal{B}$, X, $\boldsymbol{v}$)) how well $\boldsymbol{v}$ fits with the local structure of $\mathcal{B}$ around X. Based on the history of measurements $\boldsymbol{c}_X$ for node X, the method tests (using ShiftInStream($\boldsymbol{c}_X$, k)) whether a shift occurred k observations ago. k thus specifies the number of observations that are allowed to "pass" before the method should realize that a shift has taken place. We therefore call the parameter k the *allowed delay* of the method. Finally, when the actual detection has taken place, the detection algorithm invokes the updating algorithm (UpdateNet($\cdot$)) with the set of nodes for which ShiftInStream($\cdot$) detected a change, together with the last k observations.

Algorithm 1 *Algorithm for BN adaption. The algorithm takes as input an initial network $\mathcal{B}$, defined over variables $\boldsymbol{V}$, a data stream $\mathcal{D}$, and an allowed delay k for detecting shifts in $\mathcal{D}$.*

```
procedure Adapt(B, V, D, k)
    D' ← []
    c_X ← []   (∀X ∈ V)
    loop
        v ←NextCase(D)
        Append(D', (v))
        C ← ∅
        for X ∈ V do
            c ←ConflictMeasure(B, X, v)
            Append(c_X, c)
            if ShiftInStream(c_X, k) then
                C ← C ∪ {X}
        D' ←LastKEntries(D', k)
        if C ≠ ∅ then
            UpdateNet(B, C, D')
```

To monitor how well each observation $\boldsymbol{x} \equiv (x_1, \ldots, x_n)$ "fit" the current model $\mathcal{B}$, and especially the connections between a node X_i and the remaining nodes in $\mathcal{B}$, we have followed the approach of Jensen et al. (1990): If the current model is correct or is at least a good predictor of future observations, then we would in general expect that the individual elements of an observation $\boldsymbol{v}$ are positively correlated (unless $\boldsymbol{v}$ is a rare case, in which case all bets are off):

$$\log \frac{P_{\mathcal{B}}(X_i = x_i)}{P_{\mathcal{B}}(X_i = x_i | X_j = x_j \ (\forall j \neq i))} < 0 \ . \tag{3}$$

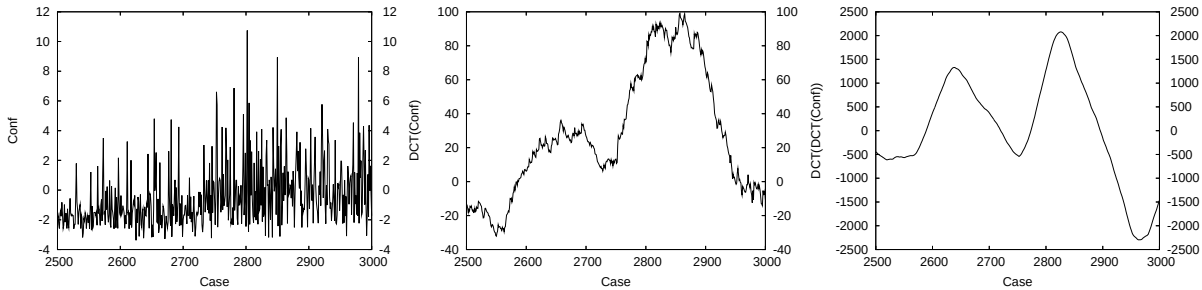
Therefore, we let ConflictMeasure($\mathcal{B}$, X_i, $\boldsymbol{v}$) return the value given on the left-hand side of (3). Note that this is where the assumption of complete data comes into play: If $\boldsymbol{v}$ is not completely observed, then (3) cannot be evaluated for all nodes X_i.

Since a high value returned by ConflictMeasure($\cdot$) for a node X could be caused by a rare case, we cannot use that value directly for determining whether a shift has occurred. This can easily be seen from the plot of values shown in Figure 1(a) in which only those following case 2725 should be considered "high". So instead we look at the block of values from before the last k cases, and compare them with the values from the last k cases. If there is a tendency towards higher values in the latter, then we conclude that this cannot be caused by only rare cases, and that a shift must have occurred. Specifically, for each variable X, ShiftInStream($\boldsymbol{c}_X$, k) check

whether there is a significant increase in the values of the last k entries in $\boldsymbol{c}_X$ relative to those before that. In our implementation we first calculate the negative of the *second discrete cosine transform* (DCT) component (see e.g. (Press et al. 2002)) of the last $2k$ measures $c_1, \ldots, c_{2k}$ in $\boldsymbol{c}_X$:

$$C_2 \equiv \sum_{j=1}^{2k} c_j \cos\left(\frac{\pi(j+\frac{1}{2})}{2k}\right) .$$

An example of such values are plotted in Figure 1(b). The component calculated after reception of each observation tells how much there is a tendency for the last $2k$ measures to be arranged on a line with a positive slope. A high value therefore indicates that the last k cases are more in conflict with the model than those from before these.



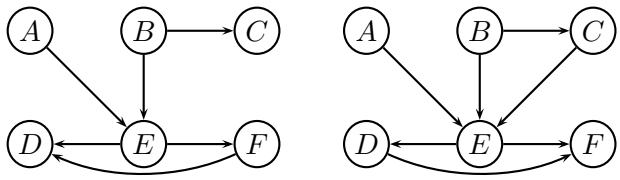
(a) (b) (c)

Figure 1: *A situation where a shift happens at observation 2725 as it is reflected in (a) the conflict measure values, (b) the DCT-component of the last 200 conflict measure values, and (c) the DCT-component of the last 100 DCT-components of the conflict measure values.*

As can be seen from Figure 1(b), the DCT statistic tends to increase, when a shift occurs, and then drop after a while. This is to be expected: As new data arrives, the conflict measures of the old well-fitting observations will be pushed out of $\boldsymbol{c}_X$, and eventually the measures in $\boldsymbol{c}_X$ will no longer be arranged on a line with strictly positive slope. We wish to react exactly when the DCT statistic starts to drop (around 2850 in the figure), since this means that the slope of the conflict measures is maximal, and hence that the last k observations at this point are more different from those before, than they would be at any other point. However, as can be seen from the figure the curve of DCT statistics is not smooth, and we might be tricked into reacting too soon. To ensure against this, we calculate the DCT-component *of the last k DCT statistics*, as seen in Figure 1(c). When this meta statistic is at 0, it means that the last k DCT statistics is approximately arranged on a line with slope 0. This should happen exactly $\frac{k}{2}$ observations after the DCT statistic reaches its maximum and start to drop steadily. In the figure, this would be a little before observation 2900.

We are unaware of any previous work using this technique for change point detection in data streams, but we chose to use this as initial experiments showed that it outperformed more traditional methods such as log-odds ratios (Nielsen and Jensen 2005) and t-tests (Press et al. 2002) in our setting.

Example 1 (A Simple Example): *Consider the two BNs in Figures 2(a) and 2(b) and imagine that the sequence of observations $\mathcal{D}$ consists of n_1 observations sampled from $\mathcal{B}_1$ and n_2 from $\mathcal{B}_2$. When feeding $\mathcal{D}$ to* ADAPT$(\cdot)$ *in Algorithm 1 starting with $\mathcal{B} = \mathcal{B}_1$ and using a value of k less than $\frac{n_1}{2}$ and n_2, the algorithm first constructs empty histories $\boldsymbol{c}_A, \ldots, \boldsymbol{c}_F$ and a list for the last k cases $\mathcal{D}'$, which is initially empty.*



(a) $\mathcal{B}_1$ (b) $\mathcal{B}_2$

Figure 2: *The two BNs we wish to adapt to in Examples 1 and 2.*

After reading each case of $\mathcal{D}$ the algorithm computes the conflict measure for each variable, and adds it to the corresponding history. After reception of the first $2k$ cases, the algorithm starts testing if there appears to be a jump in the values in a history k cases back. When the algorithm reads the k'th case of the part of $\mathcal{D}$ drawn from $\mathcal{B}_2$ a jump should be detected for nodes A, C, and E, as each has gotten a new Markov boundary in $\mathcal{B}_2$, which we expect to manifest itself in the values of the conflict measures.

Reacting to the detection, the algorithm tells UpdateNet$(\cdot)$ *to update $\mathcal{B}$ around the nodes in $\{A, B, C\}$, based on the last k cases, which at this point are all samples from $\mathcal{B}_2$.*

4.2 Learning and Incorporating Changes

Algorithm 2 *Update Algorithm for BN. Takes as input the network to be updated $\mathcal{B}$, a set of variables whose structural bindings may be wrong $\boldsymbol{C}$, and data to learn from $\mathcal{D}'$.*

procedure UpdateNet($\mathcal{B}$, $\boldsymbol{C}$, $\mathcal{D}'$)
  $\boldsymbol{G} \leftarrow \varnothing$
  for $X \in \boldsymbol{C}$ **do**
    $\mathcal{G}_X \leftarrow$LearnFragment(X, $\mathcal{D}'$, $\boldsymbol{G}$)
    $\boldsymbol{G} \leftarrow \boldsymbol{G} \cup \{\mathcal{G}_X\}$
  for $X \in \boldsymbol{V} \setminus \boldsymbol{C}$ **do**
    $\mathcal{G}_X \leftarrow$ExtractFragment(X, $\mathcal{B}$, $\boldsymbol{G}$)
    $\boldsymbol{G} \leftarrow \boldsymbol{G} \cup \{\mathcal{G}_X\}$
  $\mathcal{G}' \leftarrow$MergeFragments($\boldsymbol{G}$)
  $(\mathcal{G}'', \boldsymbol{C}') \leftarrow$Direct($\mathcal{G}'$, $\mathcal{B}$, $\boldsymbol{C}$)
  $\boldsymbol{\Phi}'' \leftarrow \varnothing$
  for $X \in \boldsymbol{V}$ **do**
    if $X \in \boldsymbol{C}'$ **then**
      $\boldsymbol{\Phi}'' \leftarrow \boldsymbol{\Phi}'' \cup \{\mathcal{D}'(X|\mathrm{pa}_{\mathcal{G}''}(X))\}$[a]
    else
      $\boldsymbol{\Phi}'' \leftarrow \boldsymbol{\Phi}'' \cup \{P_{\mathcal{B}}(X|\mathrm{pa}_{\mathcal{G}''}(X))\}$
  $\mathcal{B} \leftarrow (\mathcal{G}'', \boldsymbol{\Phi}'')$

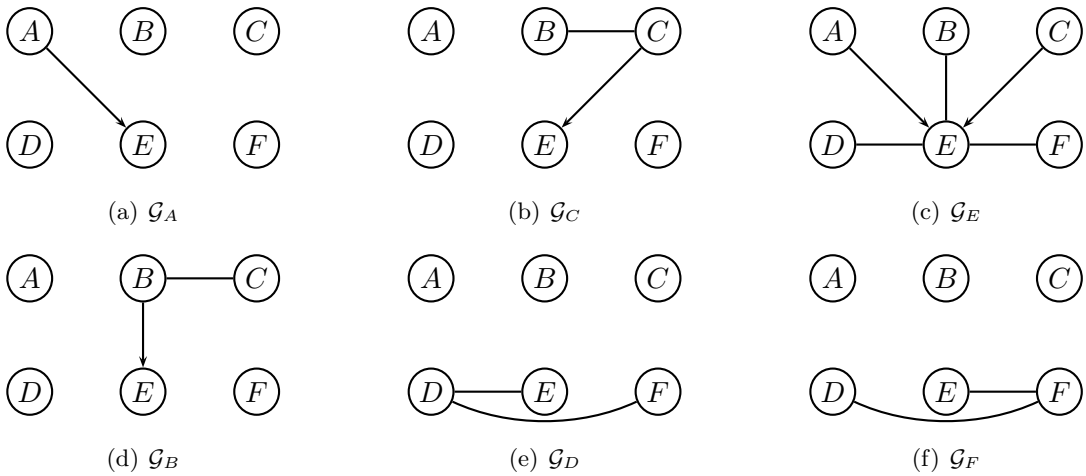
[a] In the implementation we used a Bayesian estimate rather than $\mathcal{D}'$

When a shift involving nodes $\boldsymbol{C}$ has been detected, UpdateNet($\mathcal{B}$, $\boldsymbol{C}$, $\mathcal{D}'$) in Algorithm 2 adapts the BN $\mathcal{B}$ around the nodes in $\boldsymbol{C}$ to fit the empirical distribution defined by the last k cases $\mathcal{D}'$ read from $\mathcal{D}$. Throughout the text, both the cases and the empirical distribution will be denoted $\mathcal{D}'$. Since we want to reuse the knowledge encoded in $\mathcal{B}$ that has not been deemed

outdated by the detection part of the method, we will update $\mathcal{B}$ to fit $\mathcal{D}'$ based on the assumption that only nodes in $\boldsymbol{C}$ need updating of their probabilistic bindings (i.e. the structure associated with their Markov boundaries in $\mathcal{B}_{\mathcal{D}'}$). Ignoring most details for the moment, the updating method in Algorithm 2 first runs through the nodes in $\boldsymbol{C}$ and learns a partially directed *graph fragment* $\mathcal{G}_X$ for each node X ($\mathcal{G}_X$ can roughly be thought of as a "local completed pattern" for X). When graph fragments have been constructed for all nodes in $\boldsymbol{C}$, corresponding fragments are extracted from the original graph of $\mathcal{B}$ for each of the nodes not in $\boldsymbol{C}$. All of the fragments are then merged into a single graph $\mathcal{G}'$, which is directed using four direction rules that try to preserve as much of $\mathcal{B}$'s structure as possible, without violating the newly uncovered knowledge represented by the learned graph fragments. Finally, new CPTs are constructed for those nodes $\boldsymbol{C}'$ that have gotten a new parent set in $\mathcal{B}_{\mathcal{D}'}$ (nodes which, ideally, should be a subset of $\boldsymbol{C}$). Before we describe the details related to fragment learning, extraction, and the merge and direct operations, we illustrate the main workings of the algorithms with an example.

Example 2 (A Simple Example — Part II): *Consider again the two BNs in Figures 2(a) and 2(b) and let us concentrate of the part of Example 1 where* ADAPT*(·) has called* UPDATENET*($\mathcal{B}$, $\{A, C, E\}$, $\mathcal{D}'$) in Algorithm 2. Recall that $\mathcal{D}'$ at this point consists of k observations from $\mathcal{B}_2$ and that $\mathcal{B} = \mathcal{B}_1$.*

UPDATENET*(·) first learns fragments for A, C, and then E, which are shown in Figures 3(a) to 3(c). These are all learned solely on the basis of the observations in $\mathcal{D}'$. Following this, fragments for B, D and F are extracted from $\mathcal{B}$, and they are shown in Figures 3(d) to 3(f). Of these fragments only $\mathcal{G}_B$ and $\mathcal{G}_E$ disagrees on a feature, namely the connection between B and E. When the fragments are merged (into the graph in Figure 4(a)), the arc takes precedence over the link, which will be further elaborated upon below. The algorithm ends by directing the remaining links, as shown in Figure 4(b), in a manner described in more detail below.*



(a) $\mathcal{G}_A$ (b) $\mathcal{G}_C$ (c) $\mathcal{G}_E$

(d) $\mathcal{G}_B$ (e) $\mathcal{G}_D$ (f) $\mathcal{G}_F$

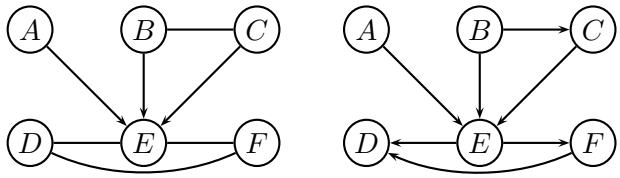
Figure 3: *Learned and extracted graph fragments matching $\mathcal{B}_2$.*

More precisely, a *graph fragment* for a node X, is a partially directed graph were each link and arc is connected to X. The fragment is intended to capture two aspects of the empirical distribution $\mathcal{D}'$: (1) The variables that can be rendered conditionally independent of X by conditioning on some set of variables are nonadjacent to X, and (2) the neighboring variables that, when added to such a conditioning set, reestablish the probabilistic connection with X

are the children of X. As we prove in (Nielsen and Nielsen 2007), having this knowledge for each variable is sufficient to establish the equivalence class of $\mathcal{B}_{\mathcal{D}'}$. This also means that an arc in a fragment $\mathcal{G}_X$ is compelled in the network produced by MergeFragments($\cdot$). A link, on the other hand, only means that the corresponding nodes must be neighbors in the network. Therefore, from the point of view of MergeFragments($\cdot$), two graph fragments $\mathcal{G}_X$ and $\mathcal{G}_Y$ are in disagreement only if either

- X and Y are nonadjacent in one fragment but adjacent in the other, or
- X and Y are connected by an arc $X \rightarrow Y$ in one fragment but $X \leftarrow Y$ in the other.

As we shall see later, the construction of graph fragments guarantee that the constructed fragments do not disagree in this manner.[4]



(a) $\mathcal{G}_2'$ (b) $\mathcal{G}_2$

Figure 4: *(a) The merged graph fragments and (b) the fully directed version.*

The actual algorithm for learning a fragment $\mathcal{G}_X$ for a changed node X is given in Algorithm 3. The algorithm consists of four main steps: In Line 2, the graph fragment is first initialized to contain the arcs and links from previously uncovered graph fragments (see Algorithm 4). Note that the connection between nodes Y and X can be found in at most one of these previously uncovered fragments (viz $\mathcal{G}_Y$), so the set of fragments trivially agrees on this connection. As per the semantics just described, arcs incorporated in this way are there to stay, but links can be turned into arcs if needed. If the previous fragments are all correct reflections of the probabilistic bindings in $\mathcal{D}'$, and this is indeed DAG faithful, then this first step should simply save some independence tests. However, in case the assumptions are violated, this step implies that fragments already learned have "precedence" over fragments learned later. In the current implementation fragments for nodes are learned in lexicographical order. Second, in Line 3, the Markov boundary of X in the empirical distribution $\mathcal{D}'$ is determined, and together with the nodes adjacent to X they constitute the nodes $\boldsymbol{R}$ that are relevant for constructing $\mathcal{G}_X$. If the previously learned graph fragments reflect genuine probabilistic bindings in $\mathcal{D}'$, we should have that $\boldsymbol{R}$ coincides with the Markov boundary of X in $\mathcal{D}'$. Third, in Lines 4 to 12, the algorithm finds the variables $\boldsymbol{S} \subseteq \boldsymbol{R}$ that can be separated from X in $\mathcal{D}'$ by conditioning on some subset of the variables in $\boldsymbol{R}$, and the variables $\boldsymbol{E}^{|X}$ that establish connection from X to a variable in $\boldsymbol{S}$. Finally, in Lines 13 to 16, arcs are added to $\mathcal{G}_X$ going from X to each node in $\boldsymbol{E}^{|X}$, and links are added between X and nodes in $\boldsymbol{R} \setminus (\boldsymbol{S} \cup \boldsymbol{E}^{|X})$. By studying the fragments in Figures 3(a) to 3(c) it is clear that they reflect the probabilistic bindings in $P_{\mathcal{B}_2}$.

[4]Note that we cannot utilize established BN combination methods like that of (Del Sagrado and Moral 2003) as both the syntax and the semantics of our graph fragments are different from those of BNs. Specifically, a fragment $\mathcal{G}_X$ need not be fully directed, a lack of an arc between two variables Y and Z different from X does not signify anything, and an arc from X to a variable Y may be needed in the final network even if it does not participate in a v-structure in $\mathcal{G}_X$ nor is part of a path leading away from a v-structure.

Algorithm 3 *Learns a graph fragment for a variable X consistent with $\mathcal{B}^{**}_{\mathcal{D}'}$ and the fragments in $\boldsymbol{G}$.*

1: **procedure** LearnFragment(X, $\mathcal{D}'$, $\boldsymbol{G}$)
2: $(\mathcal{G}_X \equiv (\boldsymbol{V}, \boldsymbol{E}_X), \boldsymbol{N}) \leftarrow$AlignWithOtherFragments($X$, $\boldsymbol{G}$)
3: $\boldsymbol{R} \leftarrow \mathrm{adj}_{\mathcal{G}_X}(X) \cup$MarkovBoundary($X$, $\mathcal{D}'$) ▷ $\boldsymbol{R}$ holds relevant nodes
4: $\boldsymbol{S} \leftarrow \boldsymbol{N} \cap \boldsymbol{R}$ ▷ $\boldsymbol{S}$ holds separable nodes
5: $\boldsymbol{E}^{|X} \leftarrow \varnothing$ ▷ $\boldsymbol{E}^{|X}$ holds dependency enabling nodes
6: **for** $Y \in \boldsymbol{R} \setminus \mathrm{adj}_{\mathcal{G}_X}(X)$ **do**
7: **for** $\boldsymbol{Z} \subseteq \boldsymbol{R} \setminus \{Y\}$ **do**
8: **if** $I_{\mathcal{D}'}(X, Y|\boldsymbol{Z})$ **then**
9: $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{Y\}$
10: **for** $Z \in \boldsymbol{R} \setminus (\{Y\} \cup \boldsymbol{S} \cup \boldsymbol{Z} \cup \boldsymbol{E}^{|X} \cup \mathrm{pa}_{\mathcal{G}_X}(X))$ **do**
11: **if** $\neg I_{\mathcal{D}'}(X, Y|\boldsymbol{Z} \cup \{Z\})$ **then**
12: $\boldsymbol{E}^{|X} \leftarrow \boldsymbol{E}^{|X} \cup \{Z\}$
13: **for** $Y \in \boldsymbol{R} \setminus \boldsymbol{S}$ **do**
14: $\boldsymbol{E}_X \leftarrow \boldsymbol{E}_X \cup \{(X, Y)\} \setminus \{(Y, X)\}$[a]
15: **if** $Y \notin \boldsymbol{E}^{|X}$ **then**
16: $\boldsymbol{E}_X \leftarrow \boldsymbol{E}_X \cup \{(Y, X)\}$
17: **return** $(\boldsymbol{V}, \boldsymbol{E}_X)$

[a]Recall that $X \rightarrow Y$ means $(X, Y) \in \boldsymbol{E}$ *and* $(Y, X) \notin \boldsymbol{E}$.

In our experimental implementation, we used the decision tree learning method of Frey et al. (2003) to find the Markov boundary of a variable X, but this choice is not essential to the workings of the method. However, both this method and LearnFragment($\cdot$) need an "independence oracle" $I_{\mathcal{D}'}$. For this we have used Pearson's χ^2 test on $\mathcal{D}'$ (see e.g. (Press et al. 2002)).

In most constraint-based learning methods, only the direction of arcs participating in v-structures are directly uncovered using independence tests, and structural rules are relied on for directing the remaining links afterwards. For the proposed method it may happen, however, that arcs that do not form v-structures in the completed pattern have to be directed through independence tests, rather than through application of structural rules afterwards. The reason is that traditional uncovering of the direction of arcs in a v-structure $X \rightarrow Y \leftarrow Z$ relies not only on knowledge that X and Y are adjacent, and that X and Z are not, but also on the knowledge that Y and Z are adjacent. At the point, where $\mathcal{G}_X$ is learned, however, knowledge of the connections among nodes adjacent to X is not known (and may be dictated by $\mathcal{D}'$ or $\mathcal{B}$), so this traditional approach is not possible. Of course these unknown connections could be uncovered from $\mathcal{D}'$ using a constraint-based algorithm, but the entire point of the method is to avoid learning the complete new network.

A graph fragment for a node X not in $\boldsymbol{C}$ is in principle the same as a learned fragment, namely a specification of the variables that can be rendered conditionally independent of X in the empirical distribution $\mathcal{D}'$, by conditioning on some set of variables, and the nodes that when added to such conditioning sets reestablish the probabilistic connection with X. But, as we assume that the probabilistic bindings defined by $\mathcal{D}'$ for nodes outside of $\boldsymbol{C}$ do not differ from those encoded in $\mathcal{B}$, we read these off the graph of $\mathcal{B}$ rather than establish them from independence tests. This is done by ExtractFragment($\cdot$) in Algorithm 5. For a given node X, the algorithm constructs a graph fragment in three steps: In Line 2, the graph fragment is

Algorithm 4 *Initializes a graph fragment for a variable X, so that it is consistent with the fragments in $\boldsymbol{G}$.*

procedure AlignWithOtherFragments(X, $\boldsymbol{G}$)
  $\boldsymbol{N} \leftarrow \varnothing$ ▷ Nonadjacent nodes
  $\boldsymbol{E}_X \leftarrow \varnothing$
  for $\mathcal{G}_Y \equiv (\boldsymbol{V}, \boldsymbol{E}_Y) \in \boldsymbol{G}$ **do**
    if $(Y, X) \in \boldsymbol{E}_Y$ **then**[a]
      $\boldsymbol{E}_X \leftarrow \boldsymbol{E}_X \cup \{(Y, X)\}$
      if $(X, Y) \in \boldsymbol{E}_Y$ **then**
        $\boldsymbol{E}_X \leftarrow \boldsymbol{E}_X \cup \{(X, Y)\}$
    else
      $\boldsymbol{N} \leftarrow \boldsymbol{N} \cup \{Y\}$
  return $((\boldsymbol{V}, \boldsymbol{E}_X), \boldsymbol{N})$

[a]Connections in already established fragments $\mathcal{G}_Y$ can only be links or arcs out of Y.

Algorithm 5 *Extracts a graph fragment for a variable X from $\mathcal{B}$, consistent with the fragments in $\boldsymbol{G}$.*

1: **procedure** ExtractFragment(X, $\mathcal{B}$, $\boldsymbol{G}$)
2: $(\mathcal{G}_X \equiv (\boldsymbol{V}, \boldsymbol{E}_X), \boldsymbol{N}) \leftarrow$AlignWithOtherFragments($X$, $\boldsymbol{G}$)
3: $\boldsymbol{E}^{|X} \leftarrow \{Z \in \mathrm{ch}_{\mathcal{B}}(X) \ : \ \exists Y \notin \mathrm{adj}_{\mathcal{B}}(X) \text{ st } Z \in \mathrm{ch}_{\mathcal{B}}(Y)\}$
4: $\boldsymbol{E}^{|X} \leftarrow \boldsymbol{E}^{|X} \cup \bigcup_{Z \in \boldsymbol{E}^{|X}} (\mathrm{de}_{\mathcal{B}}(Z) \cap \mathrm{ch}_{\mathcal{B}}(X))$
5: **for** $Z \in \mathrm{adj}_{\mathcal{B}}(X) \setminus (\mathrm{pa}_{\mathcal{G}_X}(X) \cup \boldsymbol{N})$ **do**
6: $\boldsymbol{E}_X \leftarrow \boldsymbol{E}_X \cup \{(X, Z)\} \setminus \{(Z, X)\}$
7: **if** $Z \notin \boldsymbol{E}^{|X}$ **then**
8: $\boldsymbol{E}_X \leftarrow \boldsymbol{E}_X \cup \{(Z, X)\}$
9: **return** $(\boldsymbol{V}, \boldsymbol{E}_X)$

first initialized to be consistent with previously constructed graph fragments, in the same manner as is the case for fragments that are learned. Notice that this means that learned fragments have "precedence" over reused ones. However, if our assumption that nodes outside $\boldsymbol{C}$ have not had their probabilistic bindings changed holds true, then the extracted fragments should be identical to those that would have been learned had LearnFragment($\cdot$) been used instead of ExtractFragment($\cdot$) — as we prove in (Nielsen and Nielsen 2007). In case the assumptions do not hold, the choice of constructing fragments for changed nodes prior to extracting reusable ones, is a choice of being progressive: Assume change when in doubt. A conservative attitude could be obtained by swapping Lines 3 to 5 with Lines 6 to 8 in Algorithm 2.

After initialization of the graph fragment, ExtractFragment($\cdot$) identifies those children of X in $\mathcal{B}$ where the arc from X to the child is either participating in a v-structure (Line 3), or the child is a descendant of such a child (Line 4). These children are the ones that must remain children in the adapted graph (as we prove in (Nielsen and Nielsen 2007)). Finally, in Lines 5 to 8, all relevant nodes adjacent to X in $\mathcal{B}$ are connected to X in $\mathcal{G}_X$ with either a link or an arc depending on the tests above, unless previously uncovered fragments dictate that the nodes must be nonadjacent — something that can only happen if the previously uncovered fragment was learned rather than extracted. The fragments in Figures 3(d) to 3(f) are all results of such an analysis.

Algorithm 6 *Merges a set $\boldsymbol{G}$ of graph fragments into a single graph.*

```
procedure MergeFragments(G ≡ {G_X}_{X∈V})
    E ← ∅
    for X ∈ V do
        for Y ∈ adj_{G_X}(X) do
            if X ∉ ch_{G_Y}(Y) then
                E ← E ∪ {(Y, X)}
    return (V, E)
```

When graph fragments for all nodes in $\boldsymbol{V}$ have been constructed, they are merged through a simple graph union with preference given to arcs over links in MergeFragments($\cdot$); no conflicts among orientations of arcs can happen due to the construction of LearnFragment($\cdot$) and ExtractFragment($\cdot$). Figure 4(a) show the result of merging the graph fragments in Figures 3(a) to 3(f).

Following the merge, Direct($\mathcal{G}'$, $\mathcal{B}$, $\boldsymbol{C}$) directs the remaining links in $\mathcal{G}'$ according to the following four rules:

1. (No new v-structures) If $X - Y$ is a link, $Z \to X$ is an arc, and Z and Y are nonadjacent, then direct the link $X - Y$ as $X \to Y$.

2. (No directed cycles) If $X - Y$ is a link and there is a directed path from X to Y, then direct the link $X - Y$ as $X \to Y$.

3a (Try to preserve parent sets) If Rules 1 and 2 cannot be applied, chose a link $X - Y$ at random, such that $X \in \boldsymbol{V} \setminus \boldsymbol{C}$, and direct it as in $\mathcal{B}$.

3b (Direct randomly) If Rules 1 to 3a cannot be applied, chose a link at random, and direct it randomly.

For the final graph in the example, shown in Figure 4(b), we have that both the arcs $E \to D$ and $E \to F$ are directed using Rule 1, $F \to D$ and $B \to C$ are both directed using Rule 3a.

Due to potentially flawed statistical tests, the resultant graph may contain cycles each involving at least one node in $\boldsymbol{C}$. These are eliminated by reversing only arcs connecting to at least one node in $\boldsymbol{C}$. The reversal process resembles the one used in (Margaritis and Thrun 2000): We remove all arcs connecting to nodes in $\boldsymbol{C}$ that appears in at least one cycle. We order the removed arcs according to how many cycles they appear in, and then insert them back in the graph, starting with the arcs that appear in the least number of cycles, breaking ties arbitrarily. When at some point the insertion of an arc gives rise to a cycle, we insert the arc as its reverse.

5 Formal Correctness

We have obtained a proof ensuring that under a set of assumptions our adaptation method is "correct", in the sense that when the underlying distribution generating the sequence of data changes, the algorithm reacts and changes the current BN to accurately reflect the perfect map of the new underlying distribution. The most important consequence of this result is that it makes it clear what it means for these changes to be local, probabilistically speaking; it therefore provides formal requirements on the circumstances which the heuristic for change point detection must react to. The rigorous formal treatment including proofs is presented in (Nielsen and Nielsen 2007), but we summarize the main results below.

First, we need a few definitions, on which the assumptions are based:

Definition 2. *Let P be a DAG faithful probability distribution over variables $\boldsymbol{V}$ and $X, Y \in \boldsymbol{V}$. We say that a set $\boldsymbol{M}$ is a* maximal separating set of Y from X wrt P *if*

- $\boldsymbol{M} \subseteq \mathrm{mb}_P(X) \setminus \{Y\}$,
- $Y \perp\!\!\!\perp_P X \mid \boldsymbol{M}$, *and*
- *this holds for no $\boldsymbol{X}$, where $\boldsymbol{M} \subsetneq \boldsymbol{X} \subseteq \mathrm{mb}_P(X) \setminus \{Y\}$.*

The set of all variables in $\boldsymbol{V}$, for which a maximal separating set from X wrt P exists, we denote by $\boldsymbol{S}_P^X$ (the intuition being "*s*eparable from"). For each Y in $\boldsymbol{S}_P^X$, we denote by $\boldsymbol{E}_P^{Y|X}$ the set of nodes $Z \in \mathrm{mb}_P(X)$ for which there is at least one maximal separating set $\boldsymbol{M}$ of Y from X wrt P, such that $Z \notin \boldsymbol{M}$ ("dependency *e*nabling").

The central notion that we have built our analysis on is a probabilistic one called similarity:

Definition 3. *Let P_1 and P_2 both be DAG faithful probability distributions over variables $\boldsymbol{V}$. We say that P_1 is* similar *to P_2 on the variables $\boldsymbol{I} \subseteq \boldsymbol{V}$, if we have that*

1. $\mathrm{mb}_{P_1}(X) = \mathrm{mb}_{P_2}(X)$, *for all $X \in \boldsymbol{I}$, and*
2. $\boldsymbol{S}_{P_1}^X = \boldsymbol{S}_{P_2}^X$, *for all $X \in \boldsymbol{I}$, and*
3. $\boldsymbol{E}_{P_1}^{Y|X} \setminus \boldsymbol{S}_{P_1}^X = \boldsymbol{E}_{P_2}^{Y|X} \setminus \boldsymbol{S}_{P_2}^X$, *for all $X \in \boldsymbol{I}$ and $Y \in \boldsymbol{S}_{P_1}^X$.*

Here Bullet 3 states that inseparable variables that enable dependencies between X and Y must be the same in both P_1 and P_2. Note that similarity on a set of variables $\boldsymbol{I}$ is an equivalence relation, and two distributions similar on a set of variables $\boldsymbol{I}$ are also similar on any subset of $\boldsymbol{I}$.

The notion of similarity is crucial to our results as it turns out to be a locally sufficient and necessary criteria for guaranteeing equivalence of perfect maps of two distributions:

Theorem 4. *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be perfect maps of probability distributions P_1 and P_2 both defined over variables $\boldsymbol{V}$. Then P_1 and P_2 are similar on $\boldsymbol{V}$ iff $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent.*

That is, similarity is a necessary and sufficient local criteria for equivalence. A full proof of this result and the main formal result stated next can be found in (Nielsen and Nielsen 2007); space restrictions prevent us from presenting them here:

Theorem 5. *Let BN $\mathcal{B}_1 \equiv (\mathcal{G}_1, \boldsymbol{\Phi}_1)$ and DAG faithful probability distributions P_1 and P_2 each be defined over variables $\boldsymbol{V}$. Moreover, let $\mathcal{D}$ be a DAG faithful sample sequence of size 2 and rank r, where P_1 is the distribution of the first sample and P_2 is the distribution of the last sample. Furthermore, let P_1 be similar to P_2 on $\boldsymbol{I}$, and $\mathcal{B}_2 \equiv (\mathcal{G}_2, \boldsymbol{\Phi}_2)$ be the result of running Algorithm 1 on $\mathcal{B}_1$ with cases $\mathcal{D}$ and some choice of k. If*

1. *$\mathcal{G}_1$ is equivalent to $\mathcal{G}_{P_1}$, and*
2. *$r > k$,*[5] *and*
3. ShiftInStream*$(\cdot, k)$ is true for a variable X iff $X \notin \boldsymbol{I}$ and the algorithm is currently processing the k'th sample drawn from P_2, and*

[5] Note that Bullet 5 incorporates the traditional assumption on infinite data, and that Bullet 2 is only concerned with ensuring that ShiftInStream$(\cdot)$ always has at least k cases from a new partition of $\mathcal{D}$ to detect the change.

4. *MarkovBoundary$(X, \mathcal{D}')$ returns* $\mathrm{mb}_{P_2}(X)$*, if* $\mathcal{D}'$ *consists of samples from* P_2*, and*

5. *the oracle used in* LearnFragment$(\cdot)$ *is correct,*

then $\mathcal{G}_2$ *is equivalent to* $\mathcal{G}_{P_2}$*.*

The theorem is important as it guarantees that when our method is started with a BN, which is equivalent to the perfect map of some DAG faithful distribution P (Assumption 1 in the theorem), then no matter how often P changes, as long as it remains DAG faithful, and at least $2k$ observations[6] are drawn from P between each change, then our method continues to adapt $\mathcal{B}$ to fit the distribution, with a delay of k observations. Underlying this result are four assumptions (in addition to the one described above): Assumption 2 states that any data partition must contain at least k cases; this ensures that ShiftInStream$(\cdot)$ always has at least k cases from a new partition of $\mathcal{D}$ to detect a change. Assumption 3 specifies that after a shift has taken place, all relevant variables are identified after exactly k observation. At first this assumption may seem rather restrictive, however, it is easy to see that it can be replaced by the following weaker assumption: for a variable X we have that $X \notin \boldsymbol{I}$ iff there is at least one $i \geq k$ such that ShiftInStream$(\cdot, k)$ is true when the algorithm is processing the ith case from P_2. That i should be at least as big as k ensures that once we start learning a graph fragment for X, the actual learning (Algorithm 3) is based on k cases from P_2. Moreover, the variables involved in a shift need not be identified as a set but can be found sequentially when processing the cases (thus, the algorithm has some degree of robustness w.r.t. inaccuracies in ShiftInStream). Assumption 4 requires the algorithm for finding the Markov boundary to be sound and complete (see e.g. (Peña et al. 2005)), and Assumption 5 corresponds to the standard assumption that the statistical tests are reliable (see e.g. (Spirtes et al. 1993)).

6 Experiments and Results

To investigate how our method behaves in practice, we ran a series of experiments with a fully implemented version of the method. The purpose of the experiments was to examine if the reasoning motivating the construction of our algorithm in Section 4 is sound. More specifically, we wanted to see

1. if abstaining from learning at regular intervals, but instead only react to a heuristic like the conflict measure, can result in satisfactory performance, and

2. if the ability of the algorithm to use existing knowledge in the form of extracted graph fragments makes it more robust in cases where the algorithm starts with the correct generating network.

6.1 Configuration of Algorithm

As stated in Section 4.1, we have used the conflict measure of Jensen et al. (1990) to implement the method ConflictMeasure$(\cdot)$. However, instead of simply using $P_{\mathcal{B}}(X_i = x_i)$ in the numerator of (3), we used the probability $P_u(X_i = x_i)$ and kept P_u (*but not* $\mathcal{B}$) updated to new observations through *fractional updating with fading* (Olesen et al. 1992). We did this to

[6]Here we have a requirement on $2k$ observations between each shift, rather than the k called for by Bullet 2, because after a new network is learned using the last k cases, k other cases need to be evaluated with ConflictMeasure$(\cdot)$ wrt the newly learned network, before ShiftInStream$(\cdot)$ can be relied upon again. See Section 6 for more on this issue.

ensure that even when our method fails to acknowledge changes in some parts of the network, the distribution we compare to in ConflictMeasure($\cdot$) should reflect the currently generating distribution regardless.

Moreover, we have added a "quarantine" mechanism to the main loop of the algorithm, such that each node X, which receives a new parent set in UpdateNet($\cdot$), is prevented from entering the $\boldsymbol{C}$ set in the next $2k$ iterations of Adapt($\cdot$) in Algorithm 1. This was done in order to ensure that the conflict measure history c_X consists only of conflict measures that are calculated wrt the newly learned network.[7]

6.2 Nature of Experiments

We implemented the DAG generator of Ide et al. (2004), which allows for random generation of DAGs with a given number of nodes n and a maximum induced tree width w through a Markov chain Monte Carlo process. We adapted the method to take as input a DAG $\mathcal{G}$ and a percentage p, and use these to generate a graph based on the nodes of $\mathcal{G}$, and having connections different from those in $\mathcal{G}$ between $p\%$ of the nodes and only between these. We could therefore randomly create sequences $(\mathcal{G}_1, \ldots, \mathcal{G}_m)$ of DAGs, where each DAG $\mathcal{G}_i$ differs from $\mathcal{G}_{i-1}$ only by the links among $p\%$ of nodes. The actual $p\%$ of nodes selected could be different for each DAG in the sequence.

We turned each of these sequences into a sequence of BNs

$$(\mathcal{B}_1 \equiv (\mathcal{G}_1, \boldsymbol{\Phi}_1), \ldots, \mathcal{B}_m \equiv (\mathcal{G}_m, \boldsymbol{\Phi}_m))$$

by randomly assigning a number of states between 2 and 5 to each node, and then generating CPTs for the nodes in each DAG. We used the following four methods for generating CPTs:

- *Hard*: For each node X with parents $\boldsymbol{X}$ (possibly $\varnothing$) in $\mathcal{G}_1$, we randomly generated a distribution over the states of X for each configuration $\boldsymbol{x}$ of $\boldsymbol{X}$. For each graph $\mathcal{G}_i$, where $i > 1$, we generated distributions only for nodes having different parent sets in $\mathcal{G}_{i-1}$, and simply reused the CPTs found in $\boldsymbol{\Phi}_{i-1}$ for the other nodes.

- *Medium*: The same procedure as Hard, but for each node X with parents $\boldsymbol{X}$, we ensured that the distributions $P(X|\boldsymbol{x}_j)$ and $P(X|\boldsymbol{x}_{j+1})$ generated for any two adjacent configurations $\boldsymbol{x}_j$ and $\boldsymbol{x}_{j+1}$ had a KL-distance of at least 1. This was done to ensure at least some kind of genuine probabilistic bindings between parents and child.

- *Easy*: The same procedure as Medium, but this time using a threshold of 5 for the KL-distance, attempting to ensure strong probabilistic bindings among nodes.

- *T-Hard*: Same as the Medium method for the initial DAG $\mathcal{G}_1$. For subsequent graphs $\mathcal{G}_i$ and node X having a new parent set $\boldsymbol{X}$, for each configuration $\boldsymbol{x}$ of $\boldsymbol{X}$ we did a propagation of evidence $\boldsymbol{X} = \boldsymbol{x}$ in $\mathcal{B}_{i-1}$ and used the resulting marginal distribution over X subjected to a little random noise as $P(X|\boldsymbol{x})$. Theoretically, data points drawn from networks in a sequence of BNs, where the CPTs are generated by this method, should therefore be hard to distinguish, as each new network is an approximation of the preceding one.

So, for a BN sequence $(\mathcal{B}_1 \equiv (\mathcal{G}_1, \boldsymbol{\Phi}_i), \ldots, \mathcal{B}_m \equiv (\mathcal{G}_m, \boldsymbol{\Phi}_m))$ we had the following parameters open for adjustment: the number of nodes n in each DAG $\mathcal{G}_i$, the percentage p of nodes whose

[7] In addition to these points, we have made minor modifications to the presented algorithm in the implementation, which are described in full in (Nielsen and Nielsen 2007). For exposition purposes, we have stuck with the presentation given previously, though.

graphical bindings are changed in the transition from each $\mathcal{G}_i$ to $\mathcal{G}_{i+1}$, the maximum induced tree width w of each DAG $\mathcal{G}_i$, the number m of BNs in the sequence, and the method d used for generating each $\mathbf{\Phi}_i$ (Hard, Medium, Easy, and T-Hard).

Here we shall report on eight sets of parameters, all summarized in Table 1. We generated five BN sequences for each of Settings 1 to 4 (named Sequence 1.1 to 4.5), and three of each of Settings 5 to 8 (named Sequence 5.1 to 8.3).

Table 1: *Experimental settings. n is the number of nodes in each DAG, p is the percent of nodes among which links may differ, w is the maximum induced tree width of each DAG, m is the number of DAGs in the sequence, and d is the method used to generate CPTs.*

	n	p	w	m	d		n	p	w	m	d
Setting 1	10	30%	5	15	Hard	Setting 5	20	20%	4	15	Hard
Setting 2	10	30%	5	15	Medium	Setting 6	20	20%	4	15	Medium
Setting 3	10	30%	5	15	Easy	Setting 7	20	20%	4	15	Easy
Setting 4	10	30%	5	15	T-Hard	Setting 8	20	20%	4	15	T-Hard

We generated piecewise DAG faithful sample sequences by sampling from the BN sequences. For a given BN sequence $(\mathcal{B}_1, \ldots, \mathcal{B}_m)$, we sampled s_j samples $\mathcal{D}_j \equiv (\boldsymbol{v}_1^j, \ldots, \boldsymbol{v}_{s_j}^j)$ from each BN $\mathcal{B}_j$, and concatenated them into a DAG faithful sample sequence

$$\mathcal{D} \equiv (\boldsymbol{v}_1^1, \ldots, \boldsymbol{v}_{s_1}^1, \boldsymbol{v}_1^2, \ldots, \boldsymbol{v}_{s_2}^2, \ldots, \boldsymbol{v}_1^m, \ldots, \boldsymbol{v}_{s_m}^m) \ .$$

Each sample size s_j was drawn at random from $[s^{\min}; s^{\max}]$ using a uniform distribution over this interval. The values $s^{\min}$ and $s^{\max}$ are therefore parameters that need to be specified in advance, along with the list of parameters given above. Concretely, we used $s^{\min} = 200$ and $s^{\max} = 1000$ when sampling from BN Sequences 1.1 to 4.5, and $s^{\min} = 50$ and $s^{\max} = 1000$ when sampling from Sequences 5.1 to 8.3. We sampled five sample sequences from each BN sequence, making for a total of 160 sample sequences reported on here.

Each experiment was run by starting an adaptation method on a DAG faithful sample sequence with either the correct BN $\mathcal{B}_1$ as starting network, or a randomly generated alternative network (the same for all methods). For each of these experiments we measured the deviance and efficiency of the method wrt the KL-distance.

6.3 Methods

Our algorithm requires the setting of two parameters by hand: As described in Section 4.1 we need to specify a threshold τ for the DCT-components calculated by ShiftInStream$(\cdot)$, and we need to set an α level for the oracle $I_{\mathcal{D}}$ used for independence tests. Initial experiments seemed to indicate that τ should lie somewhere between 25 and 35, so we arbitrarily decided to test our algorithm with $\tau = 27$ and $\tau = 31$. α has traditionally been set to either 0.01 or 0.05 in the literature on constraint-based learning, so we have decided to test both of these values in the experiment. We thus have four versions of our method: A with $\tau = 27$ and $\alpha = 0.01$, B with $\tau = 27$ and $\alpha = 0.05$, C with $\tau = 31$ and $\alpha = 0.01$, and D with $\tau = 31$ and $\alpha = 0.05$.

For baseline comparisons we implemented the incremental learning methods of Lam and Bacchus (1994) and Friedman and Goldszmidt (1997). Both methods learn at regular intervals from the most recent observations and from summaries of previous data either in the form of selected data statistics (Friedman and Goldszmidt 1997) or a DAG (Lam and Bacchus 1994). We limited both methods to investigating nets with a maximum of four parents for a single node to reduce running time. The baseline comparison methods are: (Lam and Bacchus 1994)

with a greedy hill-climbing search step (E), (Friedman and Goldszmidt 1997) with a greedy hill-climbing search step (F), and (Friedman and Goldszmidt 1997) with a greedy Simulated annealing search step (G).

Both of the methods in (Lam and Bacchus 1994; Friedman and Goldszmidt 1997) need to be told how often to learn, and our method needs to be told how many entries in the conflict measure histories ShiftInStream(·) should evaluate. For fairness in comparison we chose to have Methods E to G learn every kth case and ShiftInStream(·) to evaluate $2k$ entries, as this means that each of Methods A to G learns from exactly k full cases at each learning step. For k we tried two values, 100 and 300. The resulting methods we call A-100, A-300, B-100, etc.

6.4 Results and Discussion

Space restrictions prevent us from presenting the results in full, but a selection is reproduced in Tables 2 to 4, and more is presented in (Nielsen and Nielsen 2007). Each reported number is the mean of the five sample sequences drawn from the listed BN sequence. The best number for given k (100 or 300) (lowest for deviances and highest for efficiencies) is reported in either **bold** or *italics*. For each sequence we compared the best mean score achieved by Methods A to D with the best mean score achieved by Methods E to G. We performed a t-test for significant different means (significance level 0.05) on the two columns using all five sample sequences for that row, and if the test was positive, we reported the best number in bold. Due to a combination of very long evaluation times for KL-distances of larger BNs and pressing deadlines for the paper, we have neglected to evaluate the performance of Methods B and D in the experiments based on Sequences 5.1 to 8.3.

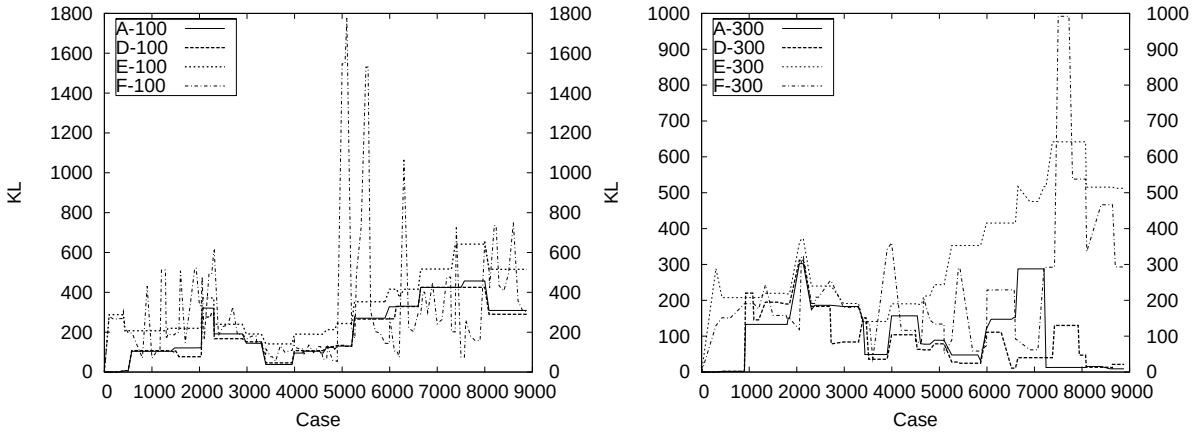
The results for deviance of the KL-distance in Table 2 show consistently poor performance for Method E, and best performance from Methods A to D. That E is performing poorly here is not so surprising, as it only remembers the past in the form of a graph structure, and not as CPTs or sets of sufficient statistics like the remaining methods.

To get a better idea of how the algorithms perform along the way, we have plotted the KL-distance between the currently learned network and the generating network for each case in one of the experiments in Figures 5(a) and 5(b). Of course, we cannot make any general statements based on this one experiment, but it is our impression that the general trend of "instability" shown by Methods E and F, compared to Methods A and D is a recurring feature in most experiments. Moreover, the tendency for some of the methods to show signs of not having converged to a steady distance at the end of the experiment tends to repeat itself in other experiments. Therefore, it could be interesting to experiment with longer BN sequences, to see how the methods perform over longer periods of time.

As would be expected, the superiority of Methods A to D on Settings 1 and 4 drops noticeably when the starting network is not the correct one (as seen in Table 3), but does not fade away completely, meaning that A to D are competitive on these cases wrt KL-distance. Finally, we can see that raising the number of nodes n to 20 and lowering the percentage p of nodes whose graphical bindings change impacts the performance of Methods E to G far worse than it does Methods A and C. We performed additional experiments were only p was changed and n was kept at 10 and these did not show the same trend, so we conclude that as the number of nodes increase Methods A to D get more attractive wrt the KL-distance. An explanation for this could be the exponential increase in search space, which does not affect Methods A to D as much as Methods E to G, since the former methods only relearn those parts of the network that seem to conflict with data, whereas the latter methods have a much looser guiding line in the form of a prior network.

Table 2: *The deviance of the KL-distance of each method when starting with the correct network averaged over five different sample sequences drawn from a BN sequence. The sequence number appears in the first column. A* ***bold*** *or italic number means that it is the lowest number for that row. A number for Methods A-D (versions of our algorithm) is reported in* ***bold*** *if it is statistically fair to say that it is lower than the lowest number for Methods E-G (baseline methods) on the same BN sequence and vice-versa.*

	A-100	B-100	C-100	D-100	E-100	F-100	G-100	A-300	B-300	C-300	D-300	E-300	F-300	G-300
1.1	124.0	**97.4**	109.0	115.0	247.0	199.0	232.0	88.7	82.7	104.0	**67.6**	241.0	158.0	215.0
1.2	101.0	81.1	94.2	**74.0**	172.0	101.0	108.0	53.0	43.0	54.1	**39.9**	167.0	89.8	82.0
1.3	57.9	52.8	66.3	**38.1**	202.0	126.0	145.0	71.5	93.4	78.7	**70.2**	197.0	111.0	118.0
1.4	149.0	*137.0*	188.0	164.0	242.0	147.0	165.0	98.8	105.0	99.8	**90.4**	237.0	136.0	150.0
1.5	116.0	119.0	90.0	**84.5**	269.0	172.0	189.0	132.0	106.0	111.0	**92.5**	260.0	124.0	129.0
2.1	57.3	60.8	58.2	**42.3**	133.0	80.3	79.5	40.2	31.5	39.8	44.4	130.0	*29.8*	50.1
2.2	161.0	153.0	174.0	*125.0*	304.0	139.0	196.0	71.3	73.0	82.2	**63.5**	294.0	131.0	170.0
2.3	137.0	151.0	**113.0**	126.0	322.0	208.0	226.0	99.8	**85.2**	122.0	102.0	315.0	153.0	185.0
2.4	81.2	75.3	54.4	**53.6**	206.0	95.8	106.0	55.4	*45.8*	56.3	45.9	194.0	55.3	60.5
2.5	97.8	87.4	83.3	118.0	167.0	**67.0**	84.5	52.6	**25.6**	40.3	36.9	160.0	43.4	63.7
3.1	227.0	215.0	216.0	195.0	366.0	*175.0*	244.0	170.0	139.0	164.0	150.0	346.0	**101.0**	232.0
3.2	250.0	277.0	280.0	258.0	587.0	**187.0**	375.0	190.0	218.0	196.0	172.0	572.0	*159.0*	269.0
3.3	43.1	**25.9**	47.5	35.1	166.0	34.3	37.5	33.2	29.9	33.1	31.0	140.0	*21.2*	31.1
3.4	28.8	26.5	27.0	30.6	90.8	*23.9*	34.7	34.4	31.5	33.8	28.3	80.1	**16.7**	26.0
3.5	225.0	203.0	214.0	224.0	406.0	*180.0*	223.0	175.0	142.0	178.0	*102.0*	397.0	119.0	180.0
4.1	*105.0*	106.0	119.0	127.0	245.0	112.0	160.0	117.0	*86.8*	95.8	96.3	241.0	115.0	121.0
4.2	126.0	134.0	135.0	**114.0**	295.0	181.0	219.0	112.0	103.0	154.0	*98.7*	289.0	115.0	158.0
4.3	104.0	103.0	102.0	*85.7*	190.0	102.0	132.0	59.3	**43.3**	53.4	54.2	187.0	82.6	85.5
4.4	116.0	83.0	119.0	**72.4**	281.0	176.0	184.0	77.9	68.9	82.5	*62.2*	273.0	74.0	135.0
4.5	118.0	104.0	91.0	*88.4*	198.0	107.0	142.0	59.1	*42.3*	70.8	52.9	190.0	64.6	97.0
5.1	120.0	N/A	**111.0**	N/A	211.0	1082.0	650.0	201.0	N/A	*196.0*	N/A	205.0	1653.0	840.0
5.2	**226.0**	N/A	227.0	N/A	275.0	1092.0	1087.0	165.0	N/A	**163.0**	N/A	267.0	600.0	652.0
5.3	263.0	N/A	**196.0**	N/A	306.0	1320.0	1127.0	235.0	N/A	**183.0**	N/A	299.0	468.0	843.0
6.1	*426.0*	N/A	466.0	N/A	433.0	2081.0	1394.0	1469.0	N/A	1678.0	N/A	**418.0**	1299.0	1117.0
6.2	421.0	N/A	*332.0*	N/A	352.0	1259.0	2433.0	329.0	N/A	*286.0*	N/A	338.0	811.0	2370.0
6.3	285.0	N/A	**194.0**	N/A	365.0	757.0	1676.0	**155.0**	N/A	213.0	N/A	350.0	1322.0	1631.0
7.1	522.0	N/A	**282.0**	N/A	355.0	727.0	956.0	277.0	N/A	**255.0**	N/A	339.0	486.0	837.0
7.2	**124.0**	N/A	135.0	N/A	149.0	383.0	458.0	**90.5**	N/A	294.0	N/A	140.0	447.0	455.0
7.3	317.0	N/A	493.0	N/A	*293.0*	483.0	1124.0	240.0	N/A	654.0	N/A	278.0	*216.0*	857.0
8.1	307.0	N/A	**291.0**	N/A	422.0	619.0	1019.0	*324.0*	N/A	2607.0	N/A	401.0	1382.0	1331.0
8.2	193.0	N/A	**180.0**	N/A	271.0	857.0	838.0	*166.0*	N/A	227.0	N/A	261.0	228.0	995.0
8.3	*279.0*	N/A	353.0	N/A	305.0	1081.0	946.0	*259.0*	N/A	446.0	N/A	293.0	1901.0	2258.0



(a) (b)

Figure 5: *KL-distance between generating distribution and learned distribution for each case in an experiment based on Sequence 1.1. To reduce clutter in the plots, we have only shown the performance of Methods A, D, E, and F.*

Table 3: *The deviance of the KL-distance of each method when starting with a network that is not correct, averaged over five different sample sequences drawn from a BN sequence. The sequence number appears in the first column. A* ***bold*** *or italic number means that it is the lowest number for that row. A number for Methods A-D (versions of our algorithm) is reported in* ***bold*** *if it is statistically fair to say that it is lower than the lowest number for Methods E-G (baseline methods) on the same BN sequence and vice-versa.*

	A-300	B-300	C-300	D-300	E-300	F-300	G-300
1.1	127.0	**81.8**	129.0	89.8	241.0	185.0	186.0
1.2	69.0	**41.9**	70.4	56.3	167.0	96.8	75.9
1.3	92.2	101.0	91.9	**69.0**	197.0	102.0	112.0
1.4	105.0	**96.7**	110.0	107.0	237.0	122.0	147.0
1.5	139.0	133.0	119.0	**101.0**	260.0	146.0	142.0
2.1	47.6	*40.1*	50.0	40.8	130.0	45.1	85.5
2.2	89.5	**71.9**	74.7	72.1	294.0	135.0	134.0
2.3	147.0	**105.0**	131.0	123.0	315.0	188.0	169.0
2.4	*47.7*	48.6	56.0	50.2	195.0	59.8	80.5
2.5	42.2	37.0	47.1	**27.8**	160.0	49.8	64.8
3.1	199.0	194.0	188.0	199.0	347.0	**90.7**	233.0
3.2	271.0	236.0	*208.0*	281.0	572.0	245.0	275.0
3.3	44.4	41.5	43.0	45.6	140.0	*32.2*	38.6
3.4	35.6	34.8	36.8	40.5	80.2	*30.2*	34.6
3.5	218.0	183.0	203.0	144.0	397.0	*94.5*	171.0
4.1	99.6	*87.6*	110.0	98.9	241.0	106.0	140.0
4.2	149.0	127.0	137.0	*122.0*	289.0	137.0	176.0
4.3	55.7	**45.6**	58.5	59.3	187.0	81.9	87.3
4.4	75.7	**48.6**	83.9	81.4	273.0	124.0	133.0
4.5	76.4	50.3	81.7	**47.3**	190.0	70.9	88.3

Table 4: *The efficiency wrt. the KL-distance of each method when starting with correct and alternative network. The first column states the sequence number. A* ***bold*** *or italic number means that it is the highest number for that row. A number for Methods A-D (versions of our algorithm) is reported in* ***bold*** *if it is statistically fair to say that it is higher than the lowest number for methods E-G (baseline methods) on the same BN sequence and vice-versa.*

	Correct starting network							Alternative starting network						
	A-300	B-300	C-300	D-300	E-300	F-300	G-300	A-300	B-300	C-300	D-300	E-300	F-300	G-300
1.1	-6.33	2.29	0.38	*5.63*	-27.2	-10.7	-7.17	26.2	17.8	25.7	**29.6**	-27.2	-8.09	-10.3
1.2	-2.93	-1.68	-2.56	*1.05*	-19.9	-4.49	-4.04	10.2	11.8	**17.5**	12.4	-22.0	-2.27	-5.66
1.3	-8.80	-3.28	-7.27	-1.18	-12.7	*3.19*	-10.8	-4.63	-5.96	5.89	**9.06**	-12.7	-3.09	-3.67
1.4	8.16	7.85	9.29	3.81	-13.9	*14.3*	-13.5	19.5	18.6	**22.5**	20.1	-12.0	-3.74	-11.6
1.5	**3.10**	-4.23	-1.03	-4.18	-32.1	-8.94	-14.5	*0.23*	-5.31	-15.5	-3.46	-32.1	-6.72	-13.1
2.1	*5.07*	3.18	-0.00	4.33	-3.07	0.32	1.68	*16.2*	10.8	7.82	8.31	-3.07	-0.92	2.60
2.2	-3.59	-6.85	8.06	3.50	-28.0	7.87	*13.8*	9.88	8.62	15.9	9.77	-28.0	6.30	*16.4*
2.3	6.35	0.60	*6.97*	-14.2	-18.9	-13.1	1.70	-14.1	-4.13	-2.85	*15.4*	-17.7	-5.70	-9.36
2.4	-6.57	-7.24	-7.74	-5.41	-13.7	*-3.08*	-4.35	**5.30**	-4.04	0.25	1.07	-12.7	-3.21	-2.21
2.5	0.74	**6.27**	-0.96	0.37	-2.36	0.31	-4.10	0.55	1.38	2.51	*3.23*	-2.22	0.39	-2.40
3.1	17.5	19.1	26.7	**38.7**	-24.8	-7.76	-28.2	46.5	**56.3**	33.7	40.5	-21.1	2.49	-19.3
3.2	-35.7	-31.1	-14.3	-20.0	-45.4	-1.73	*11.6*	-11.4	-17.2	**49.0**	15.9	-45.4	-20.0	-33.2
3.3	4.85	**6.01**	3.90	4.29	-9.88	0.65	-1.08	9.57	7.73	9.28	**10.9**	-9.88	0.58	-0.15
3.4	0.80	*3.04*	0.86	1.44	-7.11	1.63	1.33	8.47	8.17	**12.6**	7.69	-6.15	1.90	0.94
3.5	23.8	1.80	*27.3*	6.69	-12.5	4.52	-5.84	*28.3*	4.23	15.0	20.0	-12.5	1.97	-9.97
4.1	-16.6	-22.8	-8.03	-7.50	-20.4	**4.03**	-5.25	-9.05	-0.10	-8.77	2.79	-20.4	2.33	*11.8*
4.2	*3.63*	-19.2	-9.89	1.59	-29.4	-5.47	-13.1	-14.7	2.28	-5.30	**8.98**	-26.2	-7.26	-10.1
4.3	-2.73	*6.07*	4.66	0.67	-8.36	-0.77	0.59	-3.73	-2.00	-2.35	*3.51*	-7.29	-4.50	3.24
4.4	2.22	12.1	-2.53	**19.2**	-45.2	-4.05	-7.58	-9.05	*6.50*	-12.6	3.88	-45.2	-6.61	-8.41
4.5	-6.59	**16.9**	-0.63	8.89	-27.2	-5.47	-6.77	-12.8	-4.60	-1.53	*11.5*	-25.8	-0.05	-5.78

Looking at efficiency (Table 4), we see that a large portion of the mean improvements are negative for all methods, indicating that the methods tend to be destructive rather than constructive. Some of the numbers can be explained by the fact that the methods start with the correct network and proceed to change it over time. This can be verified by studying the experiments with alternative starting networks, where most numbers for Methods A to D are positive. The remaining low numbers could be caused by the fact that when the structure is changed by one of the learning methods the new CPTs are either estimated from the last k cases (Methods A to E) or from statistics which might be defined from only the last k cases or — even worse — from cases generated by networks long back in the sequence. In any case, the new CPTs might render the new network a worse approximation of the underlying distribution than the previous one. In general though, the picture is somewhat muddy. The poor performer is obviously Method E, but none of the other methods can really be said to be best.

The answers to the questions set forth in the beginning of this section, are thus as follows: First, it seems that the strategy of abstaining from learning, unless a sudden increase in conflict measure is detected, can yield satisfactory performance wrt the KL-distance. However, the same cannot be said about the structural difference measures. The second question on robustness, must be answered in the positive — at least for the KL-distance. The relatively better statistics for experiments with BNs having 20 nodes give a strong indication that, when only parts of the underlying nets are changed, and these nets contain a lot of links, then the conservative approach of keeping as much as possible is fruitful.

Finally, we have also investigated deviance and efficiency wrt structural distances, but the results are not readily interpretable, see (Nielsen and Nielsen 2007).

Conclusion

We have treated a problem of learning BNs in settings with unstable underlying dynamics. Despite the more realistic assumptions this problem has so far received little attention in the literature, which has prompted a thorough analysis and problem discussion here.

The method that we have presented addresses the problem of keeping a BN model structure updated in face of new observations. The distinguishing feature is to only learn when new observations are highly unlikely given the current model, and in those cases only change the parts of the model that are contested by evidence. Our method rests on a very solid formal basis (which is sadly too extensive to include here in full detail) and has been evaluated using carefully designed experiments. The experiments suggest that the method performs reasonably well in settings determined by unstable dynamics.

The heuristics the method use for detecting "highly unlikely" observations, and parts "contested by evidence" can be exchanged for other heuristics. Our formal results give clear guidelines on what must be expected of such heuristics, and new heuristics could therefore be evaluated in isolation by testing empirically how well they live up to these expectations.

Currently, we have a series of ideas for optimizing our method, including performing parameter adaptation on the maintained structure while monitoring for change points, and letting changes "cascade" by marking nodes adjacent to changed nodes as changed themselves. This latter idea might help our algorithm catch more changes when shifts occur. Moreover, currently we quarantine nodes with new structural bindings for $2k$ observations after a learning run, to ensure a stable history of conflict measures before we start using this to detect shifts. We might be able to avoid this by sampling new cases from the learned network to construct k artificial conflict measures immediately after learning.

In the future it would be interesting to see how a score-based approach to the local learning part of our method would perform. The problem with taking this road is that it does not seem to have any formal underpinnings or justification, as the measures the score-based approaches optimize are all defined in terms of a single underlying distribution — a difficulty which Friedman and Goldszmidt (1997) also allude to in their efforts to justify learning from data collections of varying size for local parts of the network.

Coinciding with the publication of (Nielsen and Nielsen 2006), Castillo and Gama (2006) presented a method that also builds on the idea of only learning when new data indicates the need, and more specifically a view of sequences of data much similar to our notion of piecewise DAG faithful data. The approach they present differs from the one presented here in several areas: First, they work with BNs with a distinguished *class* variable, and the classification accuracy of this variable is used as sole indicator of a shift in the underlying distribution. Second, the BNs they work with are tree augmented BN classifiers (Friedman et al. 1997), which is a subclass of BNs. Third, they learn a full model when resorting to structural learning, using a full hill-climbing search, unlike our local approach.

References

Buntine, W. (1991). Theory refinement on Bayesian networks. In B. D'Ambrosio, P. Smets, and P. Bonissone (Eds.), *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence 91*, pp. 52–60. Morgan Kaufmann Publishers.

Castillo, G. and J. Gama (2006). An adaptive prequential learning framework for Bayesian network classifiers. In J. G. Carbonell and J. Siekmann (Eds.), *Proceedings of the Tenth*

European Conference on Principles and Practice of Knowledge Discovery in Databases, Volume 4213 of *Lecture Notes in Computer Science*, pp. 67–78. Springer Verlag.

Del Sagrado, J. and S. Moral (2003). Qualitative combination of Bayesian networks. *International Journal of Intelligent Systems 18*(2), 237–249.

Frey, L., D. Fisher, I. Tsamardinos, C. F. Aliferis, and A. Statnikov (2003). Identifying Markov blankets with decision tree induction. In X. Wu and A. Tuzhilin (Eds.), *Proceedings of the Third IEEE International Conference on Data Mining*, pp. 59–66. IEEE Computer Society Press.

Friedman, N., D. Geiger, and M. Goldszmidt (1997). Bayesian network classifiers. *Machine Learning 29*(2/3), 131–163.

Friedman, N. and M. Goldszmidt (1997). Sequential update of Bayesian network structure. In D. Geiger and P. Shenoy (Eds.), *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pp. 165–174. Morgan Kaufmann Publishers.

Heckerman, D., D. Geiger, and D. M. Chickering (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning 20*(3), 197–243.

Ide, J. S., F. G. Cozman, and F. T. Ramos (2004). Generating random Bayesian networks with constraints on induced width. In R. L. de Mántaras and L. Saitta (Eds.), *Proceedings of the Sixteenth European Conference on Artificial Intelligence*, pp. 323–327. IOS Press.

Jensen, F. V., B. Chamberlain, T. Nordahl, and F. Jensen (1990). Analysis in HUGIN of data conflict. In P. Bonissone, M. Henrion, L. Kanal, and J. Lemmer (Eds.), *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pp. 519–528. Elsevier Science Publishing.

Lam, W. (1998). Bayesian network refinement via machine learning approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*(3), 240–251.

Lam, W. and F. Bacchus (1994). Using new data to refine a Bayesian network. In R. L. de Mantaras and D. Poole (Eds.), *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence 94*, pp. 383–390. Morgan Kaufmann Publishers.

Margaritis, D. and S. Thrun (2000). Bayesian network induction via local neighborhoods. In S. A. Solla, T. K. Leen, and K.-R. Müller (Eds.), *Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference*, pp. 505–511. MIT Press.

Nielsen, S. H. and T. D. Nielsen (2006). Adapting Bayes network structures to non-stationary domains. In M. Studený and J. Vomlel (Eds.), *Proceedings of the Third European Workshop on Probabilistic Graphical Models*, pp. 223–230. Action M Agency.

Nielsen, S. H. and T. D. Nielsen (2007). Adapting Bayes network structures to non-stationary domains. Technical report, Aalborg University.

Nielsen, T. D. and F. V. Jensen (2005). Alert systems for production plants: A methodology based on conflict analysis. In L. Godo (Ed.), *Proceedings of the Eighth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, Volume 3571 of *Lecture Notes in Computer Science*, pp. 76–87. Springer Verlag.

Olesen, K. G., S. L. Lauritzen, and F. V. Jensen (1992). aHUGIN: A system creating adaptive causal probabilistic networks. In D. Dubois and M. P. Wellman (Eds.), *Proceedings of the Eigth Conference on Uncertainty in Artificial Intelligence*, pp. 223–229. Morgan Kaufmann Publishers.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Representation & Reasoning. Morgan Kaufmann Publishers.

Peña, J. M., J. Björkegren, and J. Tegnér (2005). Scalable, efficient and correct learning of Markov boundaries under the faithfulness assumption. In L. Godo (Ed.), *Proceedings of the Eighth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, Volume 3571 of *Lecture Notes in Computer Science*, pp. 136–147. Springer Verlag.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (Eds.) (2002). *Numerical Recipes in C++: The Art of Scientific Computing* (2nd ed.). Cambridge University Press.

Roure, J. (2004). Incremental hill-climbing search applied to Bayesian network structure learning. In J. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi (Eds.), *Proceedings of the Eighth European Conference on Principles and Practice of Knowledge Discovery in Databases*, Volume 3202 of *Lecture Notes in Computer Science*. Springer Verlag.

Spirtes, P., C. Glymour, and R. Scheines (1993). *Causality, prediction, and search*. Springer Verlag.

Verma, T. and J. Pearl (1991). Equivalence and synthesis of causal models. In P. Bonissone, M. Henrion, L. Kanal, and J. Lemmer (Eds.), *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pp. 220–227. Elsevier Science Publishing.