Adapting Bayes Network Structures to Non-stationary Domains

Søren Holbech Nielsen and Thomas D. Nielsen
Department of Computer Science
Aalborg University
Fredrik Bajers Vej 7E
9220 Aalborg Ø, Denmark

Abstract

When an incremental structural learning method gradually modifies a Bayesian network (BN) structure to fit observations, as they are read from a database, 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 to be gradually 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 method that we present can thus be used in unknown environments, where it is not even known whether the dynamics of the environment are stable. We briefly state formal correctness results for our method, and demonstrate its feasibility experimentally.

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 for problem domains defined over discrete variables. One area of research that has seen much activity is the area of learning the structure of BNs, where probabilistic relationships for variables are discovered, or inferred, from a database of observations of these variables (main papers in learning include (Heckerman, 1998)). 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 very 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).

These papers 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, these are almost surely not fully describable by the observables and may change their behaviour instantaneously. A simple example of a situation, where it is unrealistic to expect a stationary generating process, is an industrial system, where 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 chance would be distributed differently from that representing the time before.

In this work we relax the assumption on stationary data, opting instead for 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, and thus do not try to deal with data where the data generating process changes gradually, as can happen when machinery is slowly being worn down.[1] Furthermore, we focus on domains where 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 the shifts take place).

2 Preliminaries

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.). $\equiv$ is used to denote "defined as".

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 correspondence between $\mathcal{G}$ and $\boldsymbol{\Phi}$ is enforced by requiring that $\boldsymbol{\Phi}$ consists of one CPT $P(X|\mathbf{PA}_{\mathcal{G}}(X))$ for each variable X, specifying a conditional probability distribution for X given each possible instantiation of the parents $\mathbf{PA}_{\mathcal{G}}(X)$ of X in $\mathcal{G}$. A unique joint distribution $P_{\mathcal{B}}$ over $\boldsymbol{V}$ is obtained by taking the product of all the CPTs in $\boldsymbol{\Phi}$. When it introduces no ambiguity, we shall sometimes treat $\mathcal{B}$ as synonymous with its graph $\mathcal{G}$.

Due to the construction of $P_{\mathcal{B}}$ we are guaranteed that all dependencies inherent in $P_{\mathcal{B}}$ can be read directly from $\mathcal{G}$ by use of 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

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

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. Verma and Pearl (1990) proved that equivalent graphs necessarily have the same skeleton and the same v-structures.[2] The equivalence class of graphs containing a specific graph $\mathcal{G}$ can then be uniquely represented by the partially directed graph $\mathcal{G}^*$ obtained from the skeleton of $\mathcal{G}$ by directing links that participate in a v-structure in $\mathcal{G}$ in the direction dictated by $\mathcal{G}$. $\mathcal{G}^*$ is called the *pattern* of $\mathcal{G}$. Any graph $\mathcal{G}'$, which is obtained from $\mathcal{G}^*$ by directing the 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.

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 also

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

holds, is called *DAG faithful*, and $\mathcal{B}_P$ (and sometimes just $\mathcal{G}_P$) 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* (Pearl, 1988) of X to be a set $\boldsymbol{S} \subseteq \boldsymbol{V} \setminus \{X\}$

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

[2]A triple of nodes (X, Y, Z) constitutes a *v-structure* iff X and Z are non-adjacent and both are parents of Y.

such that $X \perp\!\!\!\perp_P \boldsymbol{V} \setminus (\boldsymbol{S} \cup \{X\}) \mid \boldsymbol{S}$ and this holds for no proper subset of $\boldsymbol{S}$. It is easy to see that if P is DAG faithful, the Markov boundary of X is uniquely defined, and consists of X's parents, children, and children's parents in a perfect map of P. In the case of P being DAG faithful, we denote the Markov boundary by $\mathbf{MB}_P(X)$.

3 The Adaptation Problem

Before presenting our method for structural adaptation, we describe the problem more precisely:

We say that a sequence is *samples from a piecewise DAG faithful distribution*, if the sequence can be partitioned into sets such that each set is a database sampled from a single DAG faithful distribution, and the *rank* of the sequence is the size of the smallest such partition. Formally, let $\mathcal{D} = (\boldsymbol{d}_1, \ldots, \boldsymbol{d}_l)$ be a sequence of observations 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 $1 = i_1 < \cdots < i_m = l + 1$, such that each of $\mathcal{D}_j = (\boldsymbol{d}_{i_j}, \ldots, \boldsymbol{d}_{i_{j+1}-1})$, for $1 \leq j \leq m - 1$, is a sequence of samples from a DAG faithful distribution. The rank of the sequence is defined as $\min_j i_{j+1} - i_j$, and we say that $m - 1$ is its *size* and l its *length*. A pair of consecutive samples, $\boldsymbol{d}_i$ and $\boldsymbol{d}_{i+1}$, constitute a *shift* in $\mathcal{D}$, if there is j such that $\boldsymbol{d}_i$ is in $\mathcal{D}_j$ and $\boldsymbol{d}_{i+1}$ is in $\mathcal{D}_{j+1}$. Obviously, we can have any sequence of observations being indistinguishable from a piecewise DAG faithful sequence, by selecting the partitions small enough, 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 $i_1, \ldots, i_m$ are known.

The learning task that we address in this paper 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_t be the distribution generating sample point t. 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

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

For a method M to *adapt* to a DAG faithful sample sequence $\mathcal{D}$ wrt. *dist* then means that M seeks to minimize its deviance on $\mathcal{D}$ wrt. *dist* as possible.

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, are the same, since in such situations we re-learn 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 in the perfect maps of the two distributions changes have taken place, and only learn these parts of the new perfect map. This presents challenges, not only in detection, but also in learning the changed parts and having them fit the non-changed parts seamlessly. 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, re-learning the parts of the model that conflicts with the observations, and integrating the re-learned parts with the remaining parts of the model. These two mechanisms are described below in Sections 4.1 and 4.2, respectively.

4.1 Detecting Changes

The detection part of our method, shown in Algorithm 1, continuously processes the cases it receives. For each observation $\boldsymbol{d}$ and node X, the method measures (using ConflictMeasure($\mathcal{B}$, X, $\boldsymbol{d}$)) how well $\boldsymbol{d}$ fits with the local structure of $\mathcal{B}$ around X. Based on the history of measurements for node X, $\boldsymbol{c}_X$, the method tests (using ShiftInStream($\boldsymbol{c}_X$, k)) whether a shift occurred k observations ago. k thus acts as 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. When the actual detection has taken place, as a last step, 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. Takes as input an initial network $\mathcal{B}$, defined over variables $\boldsymbol{V}$, a series of cases $\mathcal{D}$, and an allowed delay k for detecting shifts in $\mathcal{D}$.*

```
 1: procedure Adapt($\mathcal{B}$, $\boldsymbol{V}$, $\mathcal{D}$, $k$)
 2:     $\mathcal{D}' \leftarrow []$
 3:     $\boldsymbol{c}_X \leftarrow []$   $(\forall X \in \boldsymbol{V})$
 4:     loop
 5:         $\boldsymbol{d} \leftarrow$NextCase($\mathcal{D}$)
 6:         Append($\mathcal{D}'$, $(\boldsymbol{d})$)
 7:         $\boldsymbol{S} \leftarrow \varnothing$
 8:         for $X \in \boldsymbol{V}$ do
 9:             $c \leftarrow$ConflictMeasure($\mathcal{B}$, $X$, $\boldsymbol{d}$)
10:             Append($\boldsymbol{c}_X$, $c$)
11:             if ShiftInStream($\boldsymbol{c}_X$, $k$) then
12:                 $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{X\}$
13:         $\mathcal{D}' \leftarrow$LastKEntries($\mathcal{D}'$, $k$)
14:         if $\boldsymbol{S} \neq \varnothing$ then
15:             UpdateNet($\mathcal{B}$, $\boldsymbol{S}$, $\mathcal{D}'$)
```

To monitor how well each observation $\boldsymbol{d} \equiv (d_1, \ldots, d_m)$ "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. (1991): If the current model is correct, then we would in general expect that the probability for observing $\boldsymbol{d}$ dictated by $\mathcal{B}$ is higher than or equal to that yielded by most other models. This should especially be the case for the empty model $\mathcal{E}$, where all nodes are unconnected. That is, in $\mathcal{B}$, we expect the individual attributes of $\boldsymbol{d}$ to be positively correlated (unless $\boldsymbol{d}$ is a rare case, in which case all bets are off):

$$\log \frac{P_{\mathcal{E}}(X_i = d_i)}{P_{\mathcal{B}}(X_i = d_i | X_j = d_j \ (\forall j \neq i))} > 0 \ . \qquad (1)$$

Therefore, we let ConflictMeasure($\mathcal{B}$, X_i, $\boldsymbol{d}$) return the value given on the left-hand side of (1). We note that this is where the assumption of complete data comes into play: If $\boldsymbol{d}$ is not completely observed, then (1) 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. Rather, we look at the block of values for the last k cases, and compare these with those from before that. If there is a tendency towards higher values in the former, then we conclude that this cannot be caused only by rare cases, and that a shift must have occurred. Specifically, for each variable X, ShiftInStream($\boldsymbol{c}_X$, k) checks 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 ShiftInStream($\boldsymbol{c}_X$, k) calculates the negative of the second discrete cosine transform component (see e.g. (Press et al., 2002)) of the last $2k$ measures in $\boldsymbol{c}_X$, and returns true if this statistic exceeds a pre-specified threshold value. We are unaware of any previous work using this technique for change point detection, but we chose to use this as it outperformed the more traditional methods of log-odds ratios and t-tests in our setting.

4.2 Learning and Incorporating Changes

When a shift involving nodes $\boldsymbol{S}$ has been detected, UpdateNet($\mathcal{B}$, $\boldsymbol{S}$, $\mathcal{D}'$) in Algorithm 2 adapts the BN $\mathcal{B}$ around the nodes in $\boldsymbol{S}$ 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 $\boldsymbol{S}$ 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

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 $\boldsymbol{S}$, and data to learn from $\mathcal{D}'$.*

```
1:  procedure UpdateNet($\mathcal{B}, \boldsymbol{S}, \mathcal{D}'$)
2:      for $X \in \boldsymbol{S}$ do
3:          $\widehat{\mathbf{MB}}_{\mathcal{D}'}(X) \leftarrow$MarkovBoundary($X, \mathcal{D}'$)
4:          $\mathcal{G}_X \leftarrow$Adjacencies($X, \widehat{\mathbf{MB}}_{\mathcal{D}'}(X), \mathcal{D}'$)
5:          PartiallyDirect($X, \mathcal{G}_X, \widehat{\mathbf{MB}}_{\mathcal{D}'}(X), \mathcal{D}'$)
6:      $\mathcal{G}' \leftarrow$MergeFragments($\{\mathcal{G}_X\}_{X \in \boldsymbol{S}}$)
7:      $\mathcal{G}'' \leftarrow$MergeConnections($\mathcal{B}, \mathcal{G}', \boldsymbol{S}$)
8:      $(\mathcal{G}'', \boldsymbol{C}) \leftarrow$Direct($\mathcal{G}'', \mathcal{B}, \boldsymbol{S}$)
9:      $\mathbf{\Phi}'' \leftarrow \varnothing$
10:     for $X \in \boldsymbol{V}$ do
11:         if $X \in \boldsymbol{C}$ then
12:             $\mathbf{\Phi}'' \leftarrow \mathbf{\Phi}'' \cup \{P_{\mathcal{D}'}(X|\mathbf{PA}_{\mathcal{G}''}(X))\}$
13:         else
14:             $\mathbf{\Phi}'' \leftarrow \mathbf{\Phi}'' \cup \{P_{\mathcal{B}}(X|\mathbf{PA}_{\mathcal{G}''}(X))\}$
15:     $\mathcal{B} \leftarrow (\mathcal{G}'', \mathbf{\Phi}'')$
```

in $\boldsymbol{S}$ 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 network fragments have been constructed for all nodes in $\boldsymbol{S}$, these fragments are merged into a single graph $\mathcal{G}'$, which is again merged with fragments from the original graph of $\mathcal{B}$. The merged graph is then directed using four direction rules, which 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 a new parent set in $\mathcal{B}_{\mathcal{D}'}$ (nodes which, ideally, should be a subset of $\boldsymbol{S}$).

The actual construction of $\mathcal{G}_X$ is divided into three steps: First, an estimate $\widehat{\mathbf{MB}}_{\mathcal{D}'}(X)$ of $\mathbf{MB}_{\mathcal{D}'}(X)$ is computed, using MarkovBoundary(X, $\mathcal{D}'$); second, nodes in $\widehat{\mathbf{MB}}_{\mathcal{D}'}(X)$ that are adjacent to X in $\mathcal{B}_{\mathcal{D}'}$ are uncovered, using Adjacencies(X, $\widehat{\mathbf{MB}}_{\mathcal{D}'}(X)$, $\mathcal{D}'$), and $\mathcal{G}_X$ is initialized as a graph over X and these nodes, where X is connected with links to these adjacent nodes; and third, some links in $\mathcal{G}_X$ that are arcs in $\mathcal{B}^{**}_{\mathcal{D}'}$ are directed as they would be in $\mathcal{B}^{**}_{\mathcal{D}'}$ using PartiallyDirect(X, $\mathcal{G}_X$, $\widehat{\mathbf{MB}}_{\mathcal{D}'}(X)$, $\mathcal{D}'$). See (Nielsen and Nielsen, 2006) for more elaboration on this.

In our experimental implementation, we used the decision tree learning method of Frey et al. (2003) to find $\widehat{\mathbf{MB}}_{\mathcal{D}'}(X)$, and the AlgorithmPCD($\cdot$) method in (Peña et al., 2005) (restricted to the variables in $\widehat{\mathbf{MB}}_{\mathcal{D}'}(X)$) to find variables adjacent to X. The latter method uses a greedy search for iteratively growing and shrinking the estimated set of adjacent variables until no further change takes place. Both of these methods need an "independence oracle" $I_{\mathcal{D}'}$. For this we have used a χ^2 test on $\mathcal{D}'$.

Algorithm 3 *Uncovers the direction of some arcs adjacent to a variable X as they would be in $\mathcal{B}^{**}_{\mathcal{D}'}$. $\mathbf{NE}_{\mathcal{G}_X}(X)$ consists of the nodes connected to X by a link in $\mathcal{G}_X$.*

```
1:  procedure PartiallyDirect($X, \mathcal{G}_X, \widehat{\mathbf{MB}}_{\mathcal{D}'}(X), \mathcal{D}'$)
2:      DirectAsInOtherFragments($\mathcal{G}_X$)
3:      for $Y \in \widehat{\mathbf{MB}}_{\mathcal{D}'}(X) \setminus (\mathbf{NE}_{\mathcal{G}_X}(X) \cup \mathbf{PA}_{\mathcal{G}_X}(X))$ do
4:          for $\boldsymbol{T} \subsetneq \widehat{\mathbf{MB}}_{\mathcal{D}'}(X) \setminus \{X, Y\}$ do
5:              if $I_{\mathcal{D}'}(X, Y \mid \boldsymbol{T})$ then
6:                  for $Z \in \mathbf{NE}_{\mathcal{G}_X}(X) \setminus \boldsymbol{T}$ do
7:                      if $\neg I_{\mathcal{D}'}(X, Y \mid \boldsymbol{T} \cup \{Z\})$ then
8:                          LinkToArc($\mathcal{G}_X$, $X$, $Z$)
```

The method PartiallyDirect(X, $\mathcal{G}_X$, $\widehat{\mathbf{MB}}_{\mathcal{D}'}(X)$, $\mathcal{D}'$) directs a number of links in the graph fragment $\mathcal{G}_X$ in accordance with the direction of these in (the unknown) $\mathcal{B}^{**}_{\mathcal{D}'}$. With DirectAsInOtherFragments($\mathcal{G}_X$), the procedure first exchanges links for arcs, when previously directed graph fragments unanimously dictate this. The procedure then runs through each variable Y in $\widehat{\mathbf{MB}}_{\mathcal{D}'}(X)$ not adjacent to X, finds a set of nodes in $\widehat{\mathbf{MB}}_{\mathcal{D}'}(X)$ that separate X from Y, and then tries to re-establish connection to Y by repeatedly expanding the set of separating nodes by a single node adjacent to X. If such a node can be found it has to be a child of X in the completed pattern $\mathcal{B}^{**}_{\mathcal{D}'}$, and no arc in the pattern $\mathcal{B}^{*}_{\mathcal{D}'}$ originating from X is left as a link by the procedure (see (Nielsen and Nielsen, 2006) for proofs). As before the independence oracle $I_{\mathcal{D}'}$ was implemented as a χ^2 test in our experiments.

In most constraint based learning methods, only the direction of arcs participating in v-structures are uncovered using independence tests, and structural rules are relied on for directing the remaining arcs afterwards. For the proposed method, however, more arcs from the completed pattern, than just those of v-structures, are directed through independence tests. 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 may be dictated by $\mathcal{B}$), 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 of the complete new network.

When all graph fragments for nodes in $\boldsymbol{S}$ have been constructed, they are merged through a simple graph union in MERGEFRAGMENTS($\cdot$); no conflicts among orientations can happen due to the construction of PARTIALLYDIRECT($\cdot$). In MERGECONNECTIONS($\mathcal{B}$, $\mathcal{G}'$, $\boldsymbol{S}$) connections among nodes in $\boldsymbol{V} \setminus \boldsymbol{S}$ are added according to the following rule: If $X, Y \in \boldsymbol{V} \setminus \boldsymbol{S}$ are adjacent in $\mathcal{B}$, then add the link $X - Y$ to $\mathcal{G}'$. The reason for this rule is that inseparable nodes, for which no change has been detected, are assumed to be inseparable still. However, the direction of some of the arcs may have changed in $\mathcal{G}'$, wherefore we cannot directly transfer the directions in $\mathcal{B}$ to $\mathcal{G}'$.

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

1. If $X \in \boldsymbol{V} \setminus \boldsymbol{S}$, $X - Y$ is a link, $X \rightarrow Y$ is an arc in $\mathcal{B}$, and Y is a descendant of some node Z in $\mathbf{MB}_{\mathcal{B}}(X) \setminus \mathbf{AD}_{\mathcal{B}}(X)$, where $\mathbf{AD}_{\mathcal{B}}(X)$ are nodes adjacent to X in $\mathcal{B}$, through a path involving only children of X, then direct the link $X - Y$ as $X \rightarrow Y$.[3]

2. If Rule 1 cannot be applied, and if $X - Y$ is a link, $Z \rightarrow X$ is an arc, and Z and Y are non-adjacent, then direct the link $X - Y$ as $X \rightarrow Y$.

3. If Rule 1 cannot be applied, and if $X - Y$ is a link and there is a directed path from X to Y, then direct the link $X - Y$ as $X \rightarrow Y$.

4. If Rules 1 to 3 cannot be applied, chose a link $X - Y$ at random, such that $X, Y \in \boldsymbol{V} \setminus \boldsymbol{S}$, and direct it as in $\mathcal{B}$.

5. If Rules 1 to 4 cannot be applied, chose a link at random, and direct it randomly.

Due to potentially flawed statistical tests, the resultant graph may contain cycles each involving at least one node in $\boldsymbol{S}$. These are eliminated by reversing only arcs connecting to at least one node in $\boldsymbol{S}$. The reversal process resembles the one used in (Margaritis and Thrun, 2000): We remove all arcs connecting to nodes in $\boldsymbol{S}$ 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.

We have obtained a proof of the "correctness" of the proposed method, but space restrictions prevents us from bringing it here. Basically, we have shown that, given the set-up from Section 3, if the method is started with a network equivalent to $\mathcal{B}_{P_1}$, then $\mathcal{B}_i$ will be equivalent to $\mathcal{B}_{P_{i-k}}$ for all $i > k$. This is what we refer to as "correct" behaviour, and it means that once on the right track, the method will continue to adapt to the underlying distribution, with the delay k. The assumptions behind the result, besides that each P_i is DAG faithful, are i) the samples in $\mathcal{D}$ are representative of the distributions they are drawn from, ii) the rank of $\mathcal{D}$ is bigger than $2k$, and iii) SHIFTINSTREAM($\cdot$) returns true for variable X and sample j iff P_{j-k} is not similar to P_{j-k-1} around X (see (Nielsen and Nielsen, 2006) for formal definitions and detailed proofs).

5 Experiments and Results

To investigate how our method behaves in practice, we ran a series of experiments. We constructed 100 experiments, where each consisted of five randomly generated BNs $\mathcal{B}_1, \ldots, \mathcal{B}_5$ over ten variables, each having between two and five states. We made sure that $\mathcal{B}_i$ was structurally identical to $\mathcal{B}_{i-1}$ except for the connection between two randomly chosen nodes. All CPTs in $\mathcal{B}_i$ were kept the same as in $\mathcal{B}_{i-1}$, except for the nodes with a new parent

[3]That Rule 1 is sensible is proved in (Nielsen and Nielsen, 2006). Intuitively, we try to identify a graph fragment for X in $\mathcal{B}$, that can be merged with the graph fragments learned from $\mathcal{D}'$. It turns out that the arcs directed by Rule 1 are exactly those that would have been learned by PARTIALLYDIRECT(X, $\mathcal{G}_X$, $\mathbf{MB}_{\mathcal{B}}(X)$, $P_{\mathcal{B}}$).

set. For these we employed four different methods for generating new distributions: A estimated the probabilities from the previous network with some added noise to ensure that no two distributions were the same. B, C, and D generated entirely new CPTs, with B drawing distributions from a uniform distribution over distributions. C drew distributions from the same distribution, but rejected those CPTs where there were no two parent configurations, for which the listed distributions had a KL-distance of more than 1. D was identical to C, except for having a threshold of 5. The purpose of the latter two methods is to ensure strong probabilistic dependencies for at least one parent configuration. For generation of the initial BNs we used the method of Ide et al. (). For each series of five BNs, we sampled r cases from each network and concatenated them into a piecewise DAG faithful sample sequence of rank r and length $5r$, for r being 500, 1000, 5000, and 10000.

We fed our method (NN) with the generated sequences, using different delays k (100, 500, and 1000), and measured the deciance wrt. the KL-distance on each. As mentioned we are unaware of other work geared towards non-stationary distributions, but for base-line comparison purposes, we implemented the structural adaptation methods of Friedman and Goldszmidt (1997) (FG) and Lam and Bacchus (1994) (LB). For the method of Friedman and Goldszmidt (1997) we tried both simulated annealing (FG-SA) and a more time consuming hill-climbing (FG-HC) for the unspecified search step of the algorithm. As these methods have not been developed to deal with non-stationary distributions, they have to be told the delay between learning. For this we used the same value k, that we use as delay for our own method, as this ensure that all methods store only a maximum of k full cases at any one time. The chosen k values, also correspond to those found for the experimental results reported in Friedman and Goldszmidt (1997) and Lam (1998). The only other pre-specified parameter required by our method, viz. a threshold for the χ^2-tests we set at a conventional 0.05. Each method was given the correct initial network $\mathcal{B}_1$ to start its exploration.

Space does not permit us to present the results in full, but the deviance of both NN, FG-SA, and FG-HC are presented in Figures 1 and 2. NN outperformed FG-SA in 81 of the experiments, and FG-HC in 65 of the experiments. The deviance of the LB method was much worse than for either of these three. The one experiment, where both the FG methods outperformed the NN method substantially, had r equal to 10000 and k equal to 1000, and was thus the experiment closest to the assumption on stationary distributions of the FG and LB learners.

Studying the individual experiments more closely, it became apparent that NN is more “stable” than FG: It does not alter the network as often as FG, and when doing so, NN does not alter it as much as FG. This is positive, as besides preventing unnecessary computations, it frees the user of

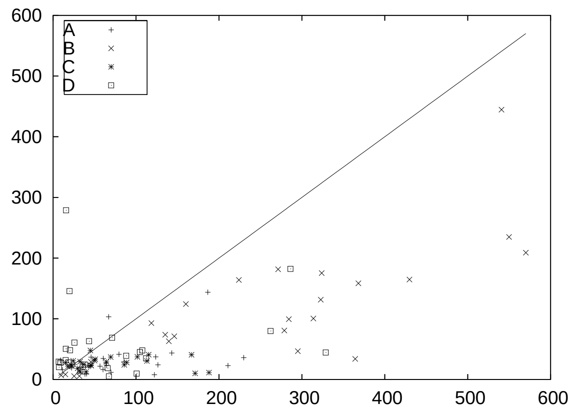

Figure 1: Deviance measures FG-SA (X-axis) vs. NN (Y-axis).

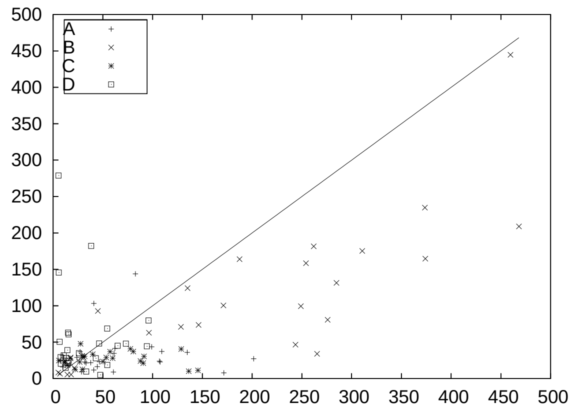

Figure 2: Deviance measures FG-HC (X-axis) vs. NN (Y-axis).

the learned nets from a range of false positives. Furthermore, we observed that the distance between the BN maintained by NN and the generating one seems to stabilize after some time. This was not always the case for FG.

6 Discussion

The plotted experiments seem to indicate that our method is superior to existing techniques for domains, where the underlying distribution is not stationary. This is especially underscored by our experiments actually favouring the existing methods through using only sequences whose rank is a multiplum of the learners k value, which means that both FG and LB always learn from data from only one partition of the sequence, unlike NN, which rarely identifies the point of change completely accurately. Moreover, score based approaches are geared towards getting small KL-scores, and thus the metric we have reported should favour FG and LB too.

Of immediate interest to us, is investigation of how our method fares when the BN given to it at the beginning is not representative of the distribution generating the first partition of the sample sequence. Also, it would be interesting to investigate the extreme cases of sequences of size 1 and those with very low ranks ($r \lll 500$). Obviously, the task of testing other parameter choices and other implementation options for the helper functions need to be carried out too.

Currently, we have some ideas for optimizing the precision of our method, including performing parameter adaptation of the CPTs associated with the maintained structure, and letting changes "cascade", by marking nodes adjacent to changed nodes as changed themselves.

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, as the measures 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.

References

W. Buntine. 1991. Theory refinement on Bayesian networks. In *UAI 91*, pages 52–60. Morgan Kaufmann.

L. Frey, D. Fisher, I. Tsamardinos, C. F. Aliferis, and A. Statnikov. 2003. Identifying Markov blankets with decision tree induction. In *ICDM 03*, pages 59–66. IEEE Computer Society Press.

N. Friedman and M. Goldszmidt. 1997. Sequential update of Bayesian network structure. In *UAI 97*, pages 165–174. Morgan Kaufmann.

D. Heckerman. 1998. A tutorial on learning with Bayesian networks. In M. Jordan, editor, *Learning in Graphical Models*, pages 301–354. Kluwer.

J. S. Ide, F. G. Cozman, and F. T. Ramos. Generating random Bayesian networks with constraints on induced width. In *ECAI 04*, pages 323–327. IOS Press.

F. V. Jensen, B. Chamberlain, T. Nordahl, and F. Jensen. 1991. Analysis in HUGIN of data conflict. In *UAI 91*. Elsevier.

W. Lam and F. Bacchus. 1994. Using new data to refine a Bayesian network. In *UAI 94*, pages 383–390. Morgan Kaufmann.

W. Lam. 1998. Bayesian network refinement via machine learning approach. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(3):240–251.

D. Margaritis and S. Thrun. 2000. Bayesian network induction via local neighborhoods. In *Advances in Neural Information Processing Systems 12*, pages 505–511. MIT Press.

S. H. Nielsen and T. D. Nielsen. 2006. Adapting bayes nets to non-stationary probability distributions. Technical report, Aalborg University. www.cs.aau.dk/~holbech/nielsennielsen_note.ps.

J. Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

J. M. Peña, J. Björkegren, and J. Tegnér. 2005. Scalable, efficient and correct learning of Markov boundaries under the faithfulness assumption. In *ECSQARU 05*, volume 3571 of *LNCS*, pages 136–147. Springer.

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

J. Roure. 2004. Incremental hill-climbing search applied to Bayesian network structure learning. In *Proceedings of the 15th European Conference on Machine Learning*. Springer.

T. Verma and J. Pearl. 1990. Equivalence and synthesis of causal models. In *UAI 91*, pages 220–227. Elsevier.