Adapting Bayes Network Structures to Non-stationary Domains — Technical Report

Søren Holbech Nielsen Thomas D. Nielsen

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 state formal correctness results for our method, and demonstrate its feasibility experimentally.

Keywords: Bayesian networks, learning, adaptation, approximations.

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, 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 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

Here we present the definitions and terminology necessary to understand 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.). $\equiv$ is used to denote "defined as", and $\leftarrow$, when used as part of algorithms, means "is set to" or "takes on the value of".

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 $P(X|\text{pa}_{\mathcal{G}}(X))$ for each variable X, specifying a conditional probability distribution for X given each possible instantiation of the parents $\text{pa}_{\mathcal{G}}(X)$ 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 $\boldsymbol{\Phi}$.

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 mean that $\{(X,Y),(Y,X)\} \subseteq \boldsymbol{E}$. In addition to $\text{pa}_{\mathcal{G}}(X)$, we introduce the notation $\text{ch}_{\mathcal{G}}(X)$, $\text{adj}_{\mathcal{G}}(X)$, $\text{de}_{\mathcal{G}}(X)$, and $\text{an}_{\mathcal{G}}(X)$ to mean the children, adjacents, descendents, and ancestors of node X in $\mathcal{G}$, respectively. When $\mathcal{G}$ is obvious from the context, we shall leave it out, and for a set of nodes $\boldsymbol{X}$, we let $\text{an}_{\mathcal{G}}(\boldsymbol{S})$ be the union $\cup_{X \in \boldsymbol{X}} \text{an}_{\mathcal{G}}(X)$. We shall often need to distinguish between different types of connections on paths between nodes: For three nodes X, Y, and Z next to each other on some path $\mathcal{X}$, we say that the connection at Y is *serial*, if arcs $X \rightarrow Y$ and $Y \rightarrow Z$ are part of $\mathcal{X}$; if arcs $X \leftarrow Y$ and $Y \rightarrow Z$ are part of $\mathcal{X}$, we call the connection *diverging*; and if arcs $X \rightarrow Y$ and $Y \leftarrow Z$ are part of $\mathcal{X}$, we call it *converging*. If for such a converging connection, we furthermore have that X and Z are not adjacent in $\mathcal{G}$, then we call the triple (X,Y,Z) a *v-structure* of $\mathcal{G}$.

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): For any path $\mathcal{X}$ between two nodes X and Y in $\mathcal{G}$, we say that $\mathcal{X}$ is *active* given a set of nodes $\boldsymbol{Z}$, if for each node W on the path, either the connection at W is serial or diverging, and W is not in $\boldsymbol{Z}$, or the connection is converging, and either W or one of its descendants are in $\boldsymbol{Z}$. If $\mathcal{X}$ is not active given $\boldsymbol{Z}$, we say that it is *blocked* by $\boldsymbol{Z}$. If all paths between two nodes X and Y are blocked by $\boldsymbol{Z}$, then we say that X and Y are *d-separated* given $\boldsymbol{Z}$. The d-separation criterion states that, if 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.

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, we refer to as that graph's *d-separation properties*. Pearl (1988) proved the following result:

Theorem 1. *Let $\mathcal{G}$ be a DAG. We have that*

$$\left\{ X \perp\!\!\!\perp_{\mathcal{G}} \boldsymbol{V} \setminus (\mathrm{de}(X) \cup \mathrm{pa}(X) \cup \{X\}) \mid \mathrm{pa}(X) \ : \ X \in \boldsymbol{V} \right\}$$

is a subset of $\mathcal{G}$'s d-separation properties.

Lauritzen et al. (1990) proved another important result:

Theorem 2. *Let $\mathcal{G}$ be a DAG. We have that $X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{Z}$ iff each path between X and Y in the ancestral graph $\mathcal{G}^m_{\mathrm{an}(\{X,Y\} \cup \boldsymbol{Z})}$ contains at least one element of $\boldsymbol{Z}$.*

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 in $\mathcal{G}$ in the direction dictated by $\mathcal{G}$. Verma and Pearl (1991) proved:

Theorem 3. *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be DAGs over $\boldsymbol{V}$. $\mathcal{G}_1$ is equivalent to $\mathcal{G}_2$ iff $\mathcal{G}_1^* = \mathcal{G}_2^*$.*

The equivalence class of graphs equivalent to a specific graph $\mathcal{G}$ can thus be uniquely represented by the pattern $\mathcal{G}^*$. 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. 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 also

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

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 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 P is a DAG faithful probability distribution 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 a perfect map of P. In those cases we shall denote the Markov boundary by $\mathrm{mb}_P(X)$, and for any DAG $\mathcal{G}$, we shall denote the set of X's parents, children, and children's parents in $\mathcal{G}$ as $\text{mb}_{\mathcal{G}}(X)$. A direct consequence of the result is that the Markov boundary of any variable in $\mathcal{G}$ must be the same in all consistent extensions of $\mathcal{G}^*$, and hence that it makes sense to speak of *the* Markov boundary for a variable in a pattern. We therefore define $\text{mb}_{\mathcal{G}^*}(X)$ to mean $\text{mb}_{\mathcal{G}}(X)$.

Given a set of variables $\boldsymbol{V}$, we call a finite multiset $\mathcal{D}$ consisting of elements of the joint state space of $\boldsymbol{V}$ a *database* over $\boldsymbol{V}$. Each element $\boldsymbol{v} \in \mathcal{D}$, we call a *case* or *observation*, and if we know the state of each $X \in \boldsymbol{V}$ corresponding to each case in $\mathcal{D}$, then we say that $\mathcal{D}$ is *complete*. If there is some total ordering over the elements in $\mathcal{D}$, then we shall call $\mathcal{D}$ a *data sequence*. For complete data and a set of variables $\boldsymbol{X} \subseteq \boldsymbol{V}$, we define the *sufficient statistics* for $\boldsymbol{X}$ wrt $\mathcal{D}$, as a potential $n_{\mathcal{D}}^{\boldsymbol{X}}(\boldsymbol{X})$, where $n_{\mathcal{D}}^{\boldsymbol{X}}(\boldsymbol{x})$ is the number of cases in $\mathcal{D}$, where all the variables in $\boldsymbol{X}$ take on the values given in $\boldsymbol{x}$. The *empirical distribution* of a complete database $\mathcal{D}$ is defined to be

$$P_{\mathcal{D}}(\boldsymbol{v}) \equiv \frac{n_{\mathcal{D}}^{\boldsymbol{V}}(\boldsymbol{v})}{|\mathcal{D}|} \,. \tag{3}$$

If each case in $\mathcal{D}$ is independently sampled from the same probability distribution P, then we say that $\mathcal{D}$ is *iid* (meaning independent and identical distributed). If P furthermore is DAG faithful, then we say that $\mathcal{D}$ contains DAG faithful data.

For any variable X with parents $\text{pa}(X)$ in a BN with DAG $\mathcal{G}$, we define the *Bayesian parameter estimate* based on $\mathcal{D}$, $P^1_{\mathcal{D},\mathcal{G}}$, as

$$P^1_{\mathcal{D},\mathcal{G}}(X = x|\text{pa}(X) = \boldsymbol{x}) = \frac{n_{\mathcal{D}}^{\{X\}\cup\text{pa}(X)}(x, \boldsymbol{x}) + 1}{n_{\mathcal{D}}^{\text{pa}(X)}(\boldsymbol{x}) + |X|} \,,$$

where $|X|$ is the number of states of 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 of $\mathcal{D}_j \equiv (\boldsymbol{v}_{i_j}, \ldots, \boldsymbol{v}_{i_{j+1}-1})$, 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, 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 $i_1, \ldots, i_{m+1}$ 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

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

For a method M to *adapt* to $\mathcal{D}$ wrt *dist* then means that M seeks to minimise its deviance on $\mathcal{D}$ wrt *dist*.

That a method aggressively adapts to piecewise DAG faithful sample sequences 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 according to the new network. 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, different from deviance, that captures 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

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

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 over time. 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. However, if two methods tend to yield comparable deviances, efficiency becomes a relevant measure.

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 in the perfect maps of the two distributions changes have taken place, and only learn these parts. This presents challenges, not only in detection, but also in learning the changed parts and having them fit the nonchanged 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, 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.

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 acts as the number of observations that are allowed to "pass" before the method should realise 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 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}
            end if
        end for
        D' ←LastKEntries(D', k)
        if C ≠ ∅ then
            UpdateNet(B, C, D')
        end if
    end loop
end procedure
```

To monitor how well each observation $\boldsymbol{v} \equiv (x_1, \ldots, x_n)$ "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 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):

$$\log \frac{P_{\mathcal{B}}(X_i = x_i)}{P_{\mathcal{B}}(X_i = x_i | X_j = x_j \ (\forall j \neq i))} < 0 \ . \tag{4}$$

Therefore, we let ConflictMeasure($\mathcal{B}$, X_i, $\boldsymbol{v}$) return the value given on the left-hand side of

(4). We note that this is where the assumption of complete data comes into play: If $\boldsymbol{v}$ is not completely observed, then (4) 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), where 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 these with those from the last k cases. If there is a tendency towards higher values in the latter, then we conclude that this cannot be caused only by rare cases, and that a shift must have occurred. Specifically, for each variable X, we have 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 $2k$ conflict measure values 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.

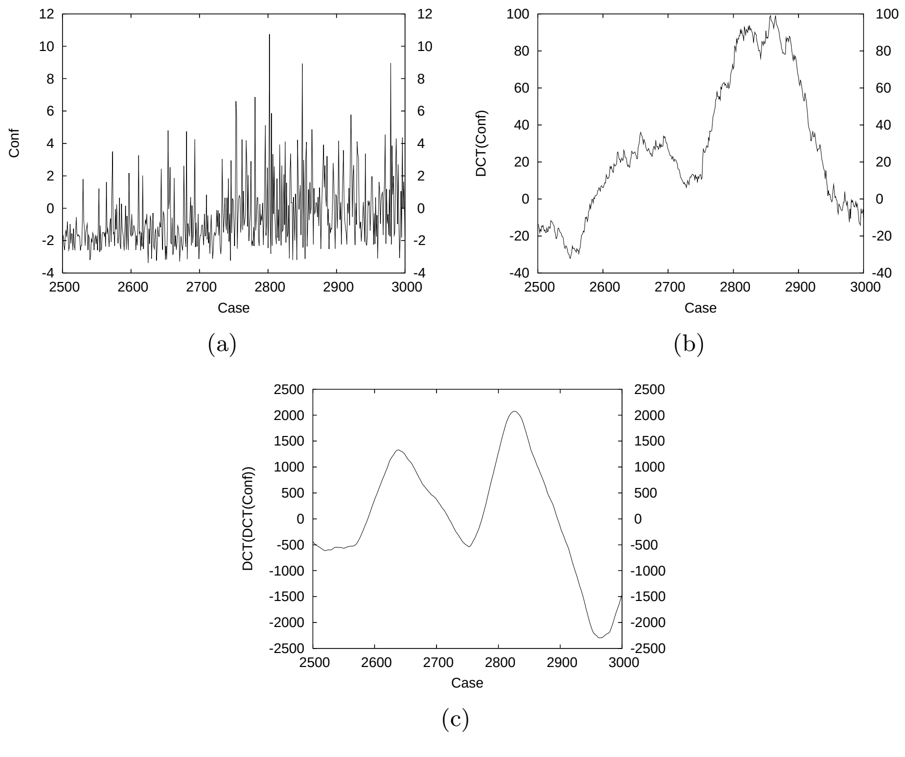
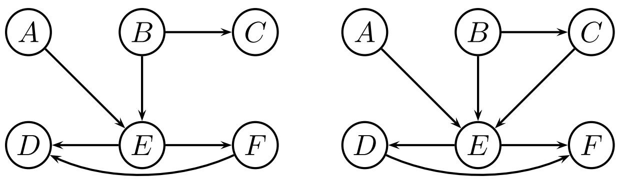


Figure 1: *A situation where a shift happens at observation 2725 as it is reflected in (a) the conflict measure values, (b) the DCT-component of the last 200 conflict measure values, and (c) the DCT-component of the last 100 DCT-components of the conflict measure values.*

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



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

Figure 2: *Two BNs from which a DAG faithful sequence has been sampled.*

of the old well-fitting observations will be pushed out of $\boldsymbol{c}_X$, and eventually the measures in this 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), as 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 it outperforming more traditional methods of 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.*

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 of the last k entries in a history from the ones prior to that. When the algorithm read 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

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

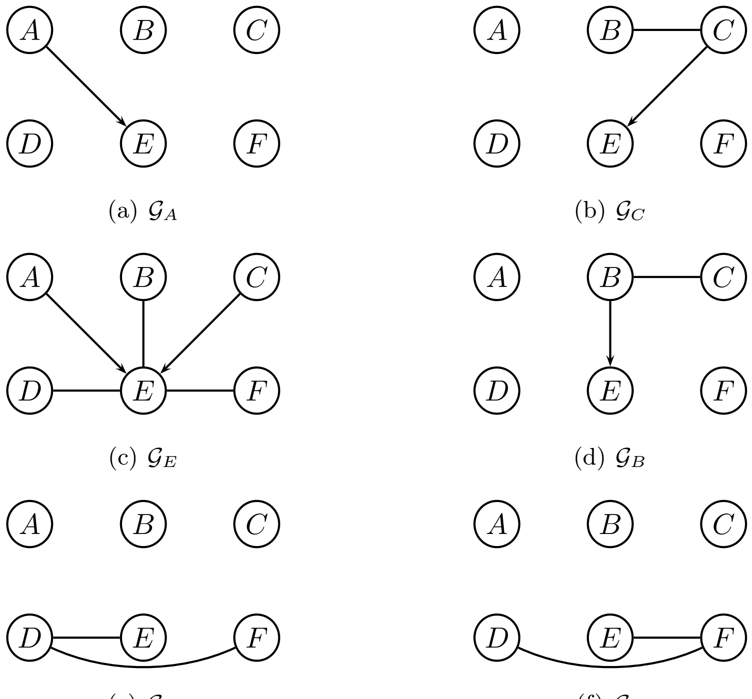
Algorithm 2 *Update Algorithm for BN. Takes as input the network to be updated $\mathcal{B}$, a set of variables whose structural bindings may be wrong $\boldsymbol{C}$, and data to learn from $\mathcal{D}'$.*

procedure UPDATENET($\mathcal{B}$, $\boldsymbol{C}$, $\mathcal{D}'$)
    $\boldsymbol{G} \leftarrow \varnothing$
    for $X \in \boldsymbol{C}$ **do**
        $\mathcal{G}_X \leftarrow$LEARNFRAGMENT(X, $\mathcal{D}'$, $\boldsymbol{G}$)
        $\boldsymbol{G} \leftarrow \boldsymbol{G} \cup \{\mathcal{G}_X\}$
    end for
    for $X \in \boldsymbol{V} \setminus \boldsymbol{C}$ **do**
        $\mathcal{G}_X \leftarrow$EXTRACTFRAGMENT(X, $\mathcal{B}$, $\boldsymbol{G}$)
        $\boldsymbol{G} \leftarrow \boldsymbol{G} \cup \{\mathcal{G}_X\}$
    end for
    $\mathcal{G}' \leftarrow$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**
            $\boldsymbol{\Phi}'' \leftarrow \boldsymbol{\Phi}'' \cup \{P^1_{\mathcal{D}',\mathcal{G}''}(X|\mathrm{pa}_{\mathcal{G}''}(X))\}$
        else
            $\boldsymbol{\Phi}'' \leftarrow \boldsymbol{\Phi}'' \cup \{P_{\mathcal{B}}(X|\mathrm{pa}_{\mathcal{G}''}(X))\}$
        end if
    end for
    $\mathcal{B} \leftarrow (\mathcal{G}'', \boldsymbol{\Phi}'')$
end procedure

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 and extraction and the merge and direct operations, we illustrate the main workings of the algorithms with an example.

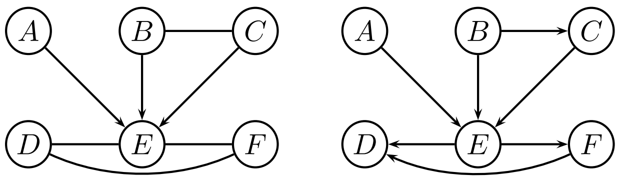
Example 2 (A Simple Example — Part II): *Consider again the two BNs in Figures 2(a) and 2(b) and let us concentrate of the part of Example 1 where* ADAPT($\cdot$) *has called* UPDATENET($\mathcal{B}$, $\{A, C, E\}$, $\mathcal{D}'$) *in Algorithm 2. Recall that $\mathcal{D}'$ at this point consists of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

UPDATENET($\cdot$) *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 $\mathcal{B}_2$.*



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

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

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 $\mathcal{D}'$: The variables that can be rendered conditionally independent of X by conditioning on some set of variables are nonadjacent to X, and the variables that when added to such conditioning sets reestablish the probabilistic connection with X are at the end of an arc if they are adjacent to X. (As we shall see in Section 5 having this knowledge for each variable is sufficient to establish the equivalence class of $\mathcal{B}_{\mathcal{D}'}$.) The more concrete meaning of an arc in a fragment $\mathcal{G}_X$ is that this arc needs to be in the network produced by MergeFragments$(\cdot)$. A link, on the other hand, only means that the node associated with X through the link must be adjacent to X in this 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 $X \to Y$ 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.

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

procedure LearnFragment(X, $\mathcal{D}'$, $\boldsymbol{G}$)
    $(\mathcal{G}_X \equiv (\boldsymbol{V}, \boldsymbol{E}_X), \boldsymbol{N}) \leftarrow$AlignWithOtherFragments($X$, $\boldsymbol{G}$)
    $\boldsymbol{R} \leftarrow \mathrm{adj}_{\mathcal{G}_X}(X) \cup$MarkovBoundary($X$, $\mathcal{D}'$) ▷ $\boldsymbol{R}$ holds relevant nodes
    $\boldsymbol{S} \leftarrow \boldsymbol{N} \cap \boldsymbol{R}$ ▷ $\boldsymbol{S}$ holds separable nodes
    $\boldsymbol{E}^{|X} \leftarrow \varnothing$ ▷ $\boldsymbol{E}^{|X}$ holds dependency enabling nodes
    for $Y \in \boldsymbol{R} \setminus \mathrm{adj}_{\mathcal{G}_X}(X)$ **do**
        for $\boldsymbol{Z} \subseteq \boldsymbol{R} \setminus \{Y\}$ **do**
            if $I_{\mathcal{D}'}(X, Y|\boldsymbol{Z})$ **then**
                $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{Y\}$
                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**
                    if $\neg I_{\mathcal{D}'}(X, Y|\boldsymbol{Z} \cup \{Z\})$ **then**
                        $\boldsymbol{E}^{|X} \leftarrow \boldsymbol{E}^{|X} \cup \{Z\}$
                    end if
                end for
            end if
        end for
    end for
    for $Y \in \boldsymbol{R} \setminus \boldsymbol{S}$ **do**
        $\boldsymbol{E}_X \leftarrow \boldsymbol{E}_X \cup \{(X, Y)\} \setminus \{(Y, X)\}$[a]
        if $Y \notin \boldsymbol{E}^{|X}$ **then**
            $\boldsymbol{E}_X \leftarrow \boldsymbol{E}_X \cup \{(Y, X)\}$
        end if
    end for
    return $(\boldsymbol{V}, \boldsymbol{E}_X)$
end procedure

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

The actual algorithm for learning a fragment $\mathcal{G}_X$ for a changed node X is given in Algorithm 3. In our experiments, we used a variation of the algorithm (given in Algorithm 7) which we describe later. For exposition purposes, however, we chose to describe the algorithm given in Algorithm 3 here. The algorithm consists of four main steps: In Line 2, the graph fragment is first initialised to contain the arcs and links from previously uncovered graph fragments. Note that the connection between a node Y and X can only be found in a maximum of 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 that are learned first 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 it together with the nodes adjacent to X constitute the nodes $\boldsymbol{R}$ that are relevant for constructing $\mathcal{G}_X$. (If the graph fragments that have previously been learned 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 17, 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 18 to 23, 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 these reflect the probabilistic bindings in $P_{\mathcal{B}_2}$.

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 uncovered using independence tests, and structural rules are relied on for directing the remaining arcs afterwards. For the proposed method, however, sometimes more arcs from the completed pattern, than only those of v-structures, has 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 $X \to Y \leftarrow Z$ 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 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 first initialised 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

Algorithm 4 *Initialises 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**
                $\boldsymbol{E}_X \leftarrow \boldsymbol{E}_X \cup \{(X, Y)\}$
            end if
        else
            $\boldsymbol{N} \leftarrow \boldsymbol{N} \cup \{Y\}$
        end if
    end for
    return $((\boldsymbol{V}, \boldsymbol{E}_X), \boldsymbol{N})$
end procedure

[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:         **end if**
10:     **end for**
11:     **return** $(\boldsymbol{V}, \boldsymbol{E}_X)$
12: **end procedure**

their probabilistic bindings changed holds true, and $\mathcal{D}'$ is DAG faithful, 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 shall prove in Section 5. 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 6 with Lines 7 to 10 in Algorithm 2.

After initialisation 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 shall see in Section 5). Finally, in Lines 5 to 10, all 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($\boldsymbol{G} \equiv \{\mathcal{G}_X\}_{X \in \boldsymbol{V}}$)
    $\boldsymbol{E} \leftarrow \varnothing$
    for $X \in \boldsymbol{V}$ **do**
        for $Y \in \mathrm{adj}_{\mathcal{G}_X}(X)$ **do**
            if $X \notin \mathrm{ch}_{\mathcal{G}_Y}(Y)$ **then**
                $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{(Y, X)\}$
            end if
        end for
    end for
    return $(\boldsymbol{V}, \boldsymbol{E})$
end procedure

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.[2] 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.

[2] We used the algorithm of Szwarcfiter and Lauer (1976) to find the cycles of DAGs.

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, and therefore provides formal requirements on the circumstances the heuristics for chance point detection must react to.

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

Definition 4. *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"). We expect, but have been unable to prove, that there is only one maximal separating set of Y from X wrt P for any DAG faithful probability distribution P:

Proposition 5. *Let P be a DAG faithful probability distribution over variables $\boldsymbol{V}$, $X \in \boldsymbol{V}$, $Y \in \boldsymbol{S}_P^X$, and $\boldsymbol{M}$ a maximal separating set of Y from X wrt P. Then $\boldsymbol{E}_P^{Y|X} = \mathrm{mb}_P(X) \setminus \boldsymbol{M}$.*

Clearly, the proposition holds true for $Y \notin \mathrm{mb}_P(X)$, as $\boldsymbol{M}$ would be $\mathrm{mb}_P(X)$. For $Y \in \mathrm{mb}_P(X)$, on the other hand, we expect that $\boldsymbol{M}$ would always be the subset of $\mathrm{mb}_P(X)$ that consists of nodes, which are not descendants of Y in any perfect map of P. The results that follow do not rely on Proposition 5 to be true, however, but we would hope that some proofs could be more concise.

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

Definition 6. *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 enabling dependencies between X and Y must be the same in both P_1 and P_2.

A few points, which should be noted, are that similarity on a set of variables $\boldsymbol{I}$ is an equivalence relation, and that 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. Since two equivalent DAGs $\mathcal{G}_1$ and $\mathcal{G}_2$ encode the same conditional independence statements, and these are reflected in any probability distributions whose perfect maps are one of $\mathcal{G}_1$ and $\mathcal{G}_2$, the following corollary is immediately obtained.

Corollary 7. *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be perfect maps of probability distributions P_1 and P_2 both defined over variables $\boldsymbol{V}$. If $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent, then P_1 and P_2 are similar on $\boldsymbol{V}$.*

That is, similarity is a necessary and local criteria for equivalence. We shall further see that it is sufficient as well (see Corollary 18), and also show that similarity has corresponding graphical characteristics.

Our main formal result, which will be established through a series of lemmas and corollaries is:

Theorem 8. *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. *$r > k$,[3] 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*
4. UNCOVERMARKOVBOUNDARY*$(X, \mathcal{D}')$ returns $\mathrm{mb}_{P_2}(X)$, if $\mathcal{D}'$ consists of samples from P_2 only, 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, then no matter how often P changes, as long as it remains DAG faithful, and at least $2k$ observations[4] 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. Especially, Bullet 3 makes it clear exactly what kind of shifts in data our heuristics should be capable of detecting.

In what follows, we gradually build up a theory to support Theorem 8, and then give the proof for it. The main difficulty is that our method directs arcs between a node and its adjacents, without knowing how these adjacents interconnect. Previous works on constraint-based DAG discovery have been based on two phase algorithms that first uncover the skeleton of the entire DAG and then direct the links into arcs. This is not possible in our case, as local parts of the graph have to be learned in isolation from the remaining graph.

To elucidate the proofs, we have added figures along the way. In all the figures we use the convention that a dashed arc or link means the existence of a path, and if the connection has

[3] 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] 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 CONFLICT-MEASURE$(\cdot)$ wrt the newly learned network, before SHIFTINSTREAM$(\cdot)$ can be relied upon again. See Section 6 for more on this issue.

an arrowhead at an end, then it means that the end of the path must be an arc oriented in that way. If the dashed connection has a label, like $\mathcal{X} \mid \boldsymbol{X}$, then it means that the name of the path is $\mathcal{X}$, and that it is known to be active given $\boldsymbol{X}$. A dotted connection labeled with $\boldsymbol{X}$, means that there is no path between the connected nodes, or that if there is such a path, then it is blocked by $\boldsymbol{X}$.

The proofs are divided into three sections: First, we show how similarity is connected with equivalence. Second, we show that the connections uncovered by LearnFragments($\cdot$) are correct wrt the empirical distribution the tests are performed on, and that the connections extracted by ExtractFragments($\cdot$) are equivalent to those that would be learned by LearnFragments($\cdot$), if it was fed data from a distribution with the current network as its perfect map. These are soundness results, and we follow them by completeness results that state that LearnFragments($\cdot$) and ExtractFragments($\cdot$) uncover all the arcs that must be identified to uniquely determine a perfect map of the underlying distribution. Finally, we use the notion of similarity, its connection to equivalence and the correctness of LearnFragments($\cdot$) and ExtractFragments($\cdot$) to prove the main theorem, acknowledging that both MergeFragments($\cdot$) and Direct($\cdot$) are almost trivially seen to be correct.

5.1 Similarity and Equivalence

We shall first prove that any two probability distributions similar on all variables, must have perfect maps with the same skeleton. For this, we need a result that we have been unable to find proved elsewhere: If two nodes can be separated by a set of variables, then they can be separated by a subset of the Markov boundary of any one of the variables too.

Lemma 9. *Let P be a DAG faithful probability distribution over $\boldsymbol{V}$ and $X, Y \in \boldsymbol{V}$. If there is a set $\boldsymbol{Y} \subseteq \boldsymbol{V} \setminus \{X, Y\}$ such that $Y \perp\!\!\!\perp_P X \mid \boldsymbol{Y}$, then there is a set $\boldsymbol{X} \subseteq \mathrm{mb}_P(X) \setminus \{Y\}$, such that $Y \perp\!\!\!\perp_P X \mid \boldsymbol{X}$.*

Proof. We assume the lemma is false, and arrive at a contradiction. Let $\mathcal{G}$ be a perfect map of P, and first note that if there is an arc between X and Y in $\mathcal{G}$, then there is no set $\boldsymbol{Y} \subseteq \boldsymbol{V} \setminus \{X, Y\}$, such that $X \perp\!\!\!\perp_P Y \mid \boldsymbol{Y}$. Furthermore, by the definition of $\mathrm{mb}_P(X)$ we have that $Y \perp\!\!\!\perp_P X \mid \mathrm{mb}_P(X)$ unless Y is a member of $\mathrm{mb}_P(X)$. The only way in which the lemma can be false, is therefore if $Y \in \mathrm{mb}_{\mathcal{G}}(X)$, X and Y are nonadjacent in $\mathcal{G}$, and there is no $\boldsymbol{X} \subseteq \mathrm{mb}_P(X) \setminus \{Y\}$, such that $Y \perp\!\!\!\perp_P X \mid \boldsymbol{X}$. As we know from Theorem 1 that the parents of X in $\mathcal{G}$ d-separates X from all nondescendants, we furthermore have that Y must be a descendant of X in $\mathcal{G}$.

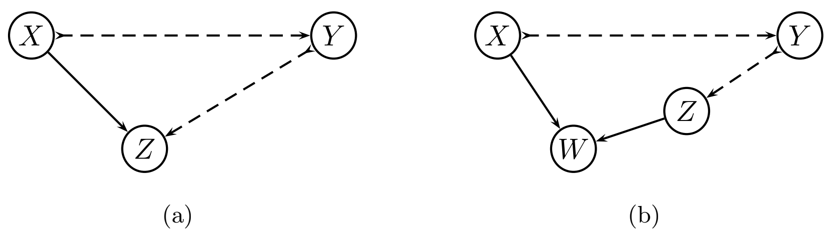


Figure 5: *Two of the possible scenarios.*

Consider the set $\boldsymbol{X}$ of all nodes in $\mathrm{mb}_{\mathcal{G}}(X)$ that are not descendants of Y in $\mathcal{G}$. If the lemma is false, $\boldsymbol{X}$ cannot d-separate X from Y in $\mathcal{G}$, and from Theorem 2 we therefore know that at least one path $\mathcal{X}$ between X and Y in $\mathcal{G}^m_{\mathrm{an}(\{X,Y\} \cup \boldsymbol{X})}$ does not contain any nodes from $\boldsymbol{X}$. Let Z be the node next to X on $\mathcal{X}$. Since Z is adjacent to X in $\mathcal{G}^m_{\mathrm{an}(\{X,Y\} \cup \boldsymbol{X})}$, it must either be (i) a parent of X in $\mathcal{G}$, (ii) a child of X, or (iii) share a child W with X. Any way, Z must be in $\mathrm{mb}_{\mathcal{G}}(X)$, and since $Z \notin \boldsymbol{X}$, Z must be a descendant of Y. We show that each of the options (i)–(iii) are impossible, and hence that there can be no $\mathcal{X}$, and consequently that the lemma must be true.

i) Z cannot be a parent of X, as it is a descendant of Y, Y is a descendant of X, and $\mathcal{G}$ is acyclic.

ii) Assume Z is a child of X in $\mathcal{G}$ (see Figure 5(a)). Since Z is in $\mathcal{G}^m_{\mathrm{an}(\{X,Y\}\cup\boldsymbol{X})}$, it must also be an ancestor of X, Y, or a node in $\boldsymbol{X}$. Since $\mathcal{G}$ is acyclic and Z is a descendant of Y, Z cannot be an ancestor of Y. Since Y is a descendant of X and Z is a descendant of Y, it can furthermore not be the case that Z is an ancestor of X. Thus, Z must be an ancestor of some node in $\boldsymbol{X}$. But these were defined as nondescendants of Y, which contradicts that Z itself is a descendant of Y.

iii) If Z shares a child W with X (see Figure 5(b)), then both Z and W are in $\mathrm{mb}_{\mathcal{G}}(X)$, and since Z is not in $\boldsymbol{X}$, it means that it is a descendant of Y, and consequently that W is a descendant of Y. This means that W is not in $\boldsymbol{X}$, a child of X, and in $\mathcal{G}^m_{\mathrm{an}(\{X,Y\}\cup\boldsymbol{X})}$. But as we saw in (ii), this is impossible. ☐

Corollary 10. *Let P be a DAG faithful probability distribution over $\boldsymbol{V}$, and let $X \not\perp\!\!\!\perp_P Y \mid \boldsymbol{X}$ for $X, Y \in \boldsymbol{V}$ and all $\boldsymbol{X} \subseteq \mathrm{mb}_P(X) \setminus \{Y\}$. Then $Y \not\perp\!\!\!\perp_P X \mid \boldsymbol{Y}$ for all $\boldsymbol{Y} \subseteq \boldsymbol{V} \setminus \{X, Y\}$.*

Since perfect maps of a distribution P are equivalent and equivalent DAGs have the same skeleton, it follows from Lemma 9 that

Corollary 11. *Let P be a DAG faithful probability distribution over $\boldsymbol{V}$, $X, Y \in \boldsymbol{V}$, and $\mathcal{G}$ be a perfect map of P. Then $Y \notin \boldsymbol{S}^X_P$ iff $Y \in \mathrm{adj}_{\mathcal{G}}(X)$.*

Given the above result it is thus sufficient for a learning algorithm to be able to identify $\boldsymbol{S}^X_P$ for all X when learning the skeleton of $\mathcal{G}_P$. Clearly, this can be seen as a localised operation. In fact, this very approach is taken by the algorithm in (Margaritis and Thrun 2000), but unfortunately the correctness proof in that paper is flawed. Specifically, Corollary 10 is claimed to be an obvious consequence of the following assumed fact: If Y is in $\mathrm{mb}_{\mathcal{G}}(X) \setminus \mathrm{adj}_{\mathcal{G}}(X)$, then any set $\boldsymbol{X}$ including $\mathrm{pa}_{\mathcal{G}}(X)$, but not including any children of Y, will d-separate X and Y in $\mathcal{G}$. Whether the result is an obvious consequence we do not speculate on, but the presented fact is not true in general, which can be seen from the graph in Figure 6, where A and B are not d-separated by the set consisting only of D, even though it satisfies the requirements.

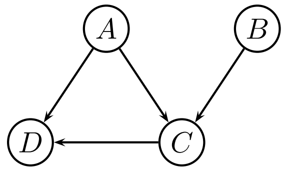


Figure 6: *A and B are not d-separated by D.*

Next, we will show that if two probability distributions are similar on all variables, then their perfect maps must have the same v-structures. For this we shall work with a subset of the arcs that are shared by perfect maps of such two distributions:

Definition 12 (Locally Compelled Arc)**.** *Let $\mathcal{G}_1$ be a perfect map of a probability distribution P_1 over $\boldsymbol{V}$, and $X, Y \in \boldsymbol{V}$, such that X is a parent of Y in $\mathcal{G}_1$. We call the arc from X to Y* locally compelled *in $\mathcal{G}_1$, if for all probability distributions P_2 similar to P_1 on $\{X\}$, and all perfect maps $\mathcal{G}_2$ of P_2, we have that X is a parent of Y in $\mathcal{G}_2$. We denote by $\mathrm{ch}^*_{\mathcal{G}}(X)$ the variables that are attached to X in $\mathcal{G}$ by locally compelled arcs originating from X.*

Locally compelled arcs are thus graphical concepts defined by similarity on *individual* variables, rather than the set of all variables of the graph seen as a whole, as is the case for compelled arcs. From Corollary 7 we can immediately see that a locally compelled arc must be compelled as well. We shall now show how locally compelled arcs include all arcs in v-structures, and thereby obtain that the set of all locally compelled arcs in a graph along with its skeleton is a unique representative of the equivalence class of the graph.

We shall need a lemma that guarantees the existence of a special subgraph in a given graph, when two simple independence statements are known to hold for nodes in the graph.

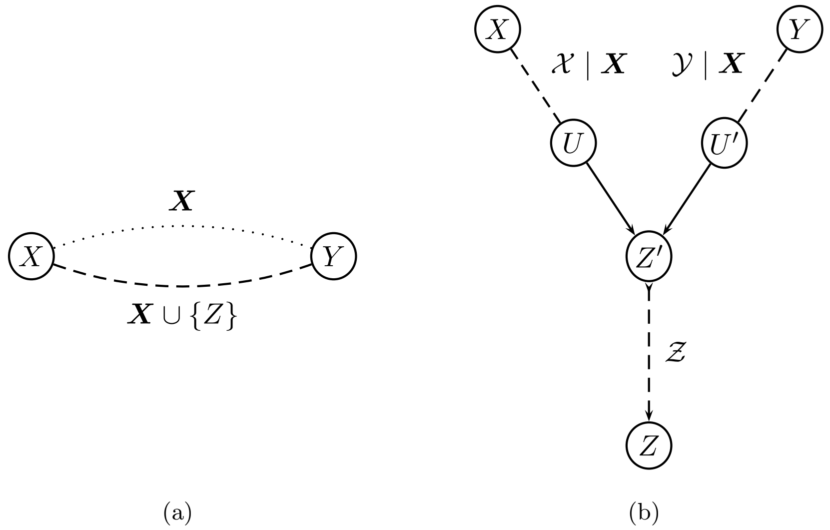


Figure 7: *(a) implies (b).*

Lemma 13. *Let $\mathcal{G}$ be a DAG over variables $\boldsymbol{V}$, $X, Y, Z \in \boldsymbol{V}$, and $\boldsymbol{X} \subseteq \boldsymbol{V} \setminus \{X, Y, Z\}$. If $X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X}$ and $X \not\!\perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X} \cup \{Z\}$ (see Figure 7(a)), then*

1. *there is a node Z' in $\mathcal{G}$ (see Figure 7(b)), with nonadjacent parents U and U', such that $Z = Z'$ or Z is a descendant of Z' on a path involving no nodes in $\boldsymbol{X}$, and*
2. *U is connected to X with a path that is active given $\boldsymbol{X}$, and*
3. *and U' is connected to Y with a path that is active given $\boldsymbol{X}$.*

Trivially, we also have $X \not\!\perp\!\!\!\perp_{\mathcal{G}} Z \mid \boldsymbol{X}$ and $Y \not\!\perp\!\!\!\perp_{\mathcal{G}} Z \mid \boldsymbol{X}$.

Proof. From the definition of d-separation, it follows from $X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X}$ and $X \not\!\perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X} \cup \{Z\}$ that there is a node Z', such that either Z is Z' or Z is a descendant of Z' along a path $\mathcal{Z}$ involving no nodes in $\boldsymbol{X}$, and Z' has two parents U and U', where U (U') is either X (Y) or connected to X (Y) on a path $\mathcal{X}$ ($\mathcal{Y}$) that is active given $\boldsymbol{X} \cup \{Z\}$. We need to show that for at least one such Z',

1. both $\mathcal{X}$ and $\mathcal{Y}$ are active given $\boldsymbol{X}$ alone, and

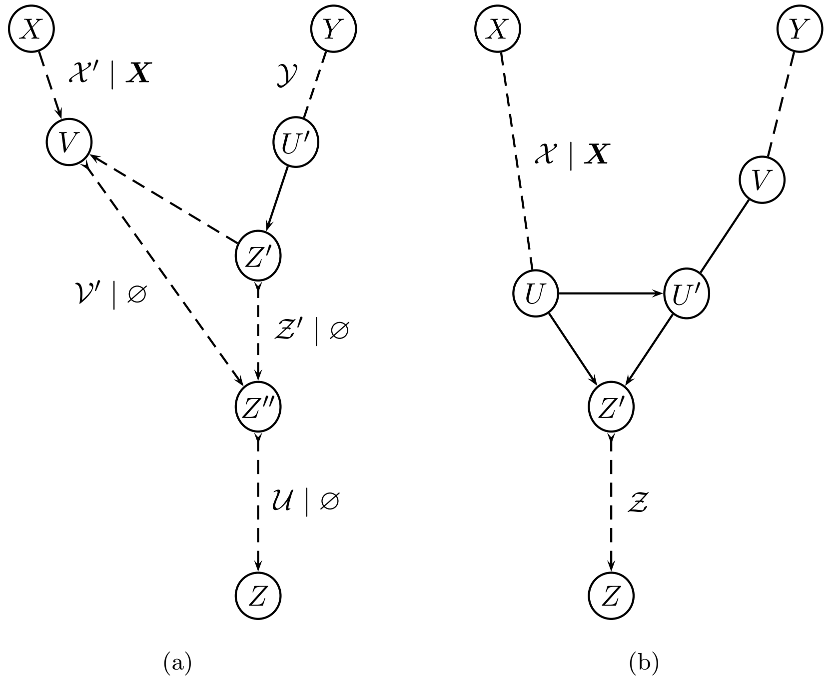


Figure 8: *(a) the transformation that removes Z from the conditioning set, and (b) the transformation that finds nonadjacent U and U'.*

2. that U and U' are not adjacent.

We first show that there is a Z' fulfilling 1, and then that at least one of these Z's fulfills 2.

Take Z' such that $\mathcal{X}$ is active given $\boldsymbol{X} \cup \{Z\}$ but not $\boldsymbol{X}$. This means that there is a converging connection at some node V on $\mathcal{X}$, such that Z is V or a descendant of V (see Figure 8(a)). Without loss of generality (WLOG) let V be chosen closest to X on $\mathcal{X}$, and let $\mathcal{X}'$ be the part of $\mathcal{X}$ from X to V, and $\mathcal{V}$ be the part from V to Z. Moreover, let Z'' be the node furthest away from Z on $\mathcal{V}$ such that Z'' is also on $\mathcal{Z}$, and let $\mathcal{U}$ be the part of $\mathcal{Z}$ from Z'' to Z, $\mathcal{V}'$ be the part of $\mathcal{V}$ from V to Z'', and $\mathcal{Z}'$ the part of $\mathcal{Z}$ from Z' to Z''. Then Z'' satisfies the definition of Z', since the three paths $\mathcal{X}'\mathcal{V}'$, $\mathcal{U}$, and $\mathcal{Z}'(Z' \leftarrow U')\mathcal{Y}$ connect X to Z'', Z'' to Z, and Z'' to Y, and the connection at Z'' between $\mathcal{X}'\mathcal{V}'$ and $\mathcal{Z}'(Z' \leftarrow U')\mathcal{Y}$ is converging. However, unlike $\mathcal{X}$, the path $\mathcal{X}'\mathcal{V}'$ is active given $\boldsymbol{X}$ only. If $\mathcal{Y}$ happens to be active only given $\boldsymbol{X} \cup \{Z\}$, we can perform the same transformation to that path and thus we get that 1 is satisfiable.

Next, we show that given a Z' that satisfies 1, we can find one that also satisfies 2. Particularly, let Z' satisfy 1, and furthermore let Z' be chosen such that $\mathcal{Z}$ has maximal length among all valid choices of Z' (see Figure 8(b)). Then U and U' must be nonadjacent. Suppose this is not the case, and WLOG let the arc between U and U' be oriented $U \to U'$. This means that U' cannot be Y as that would create a path from X to Y active given $\boldsymbol{X}$ only, contradicting that $X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X}$. So consider the arc between U' and the next node V on $\mathcal{Y}$. This arc cannot be directed into U' as that would mean that U' is a valid choice for Z', yet with the length of the path from U' to Z through Z' being longer than $\mathcal{Z}$, which contradicts the choice of Z'. But if the arc between U' and V is directed away from U' then we have that $\mathcal{X}(U \to U')\mathcal{Y}$ is an active path given $\boldsymbol{X}$, which again contradicts $X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X}$. Thus U and U' are nonadjacent. □

Notice that if $Z \in \boldsymbol{E}_P^{Y|X}$ for some X and Y, then the two independence statements $X \perp\!\!\!\perp_P Y \mid \boldsymbol{X}$

and $X \not\perp\!\!\!\perp_P Y \mid \boldsymbol{X} \cup \{Z\}$ are satisfied for some $\boldsymbol{X}$. Moreover, this must be true for each distribution similar to P on a set including X. Hence, a subgraph like the one in Figure 7(b) can be found in all perfect maps of such distributions. If we assume that Z is adjacent to X, then we can be even more precise about the connectivity of Z, though:

Lemma 14. *Let $\mathcal{G}$ be a DAG over $\boldsymbol{V}$, $X, Y \in \boldsymbol{V}$, $\boldsymbol{X} \subseteq \boldsymbol{V} \setminus \{X, Y, Z\}$, and $Z \in \mathrm{adj}_{\mathcal{G}}(X)$. If $X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X}$ and $X \not\perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X} \cup \{Z\}$, then $Z \in \mathrm{ch}_{\mathcal{G}}(X)$.*

Proof. As Z is in $\mathrm{adj}_{\mathcal{G}}(X)$, all that is left to be shown is that Z cannot be a parent of X. Assume this is false. As $X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X}$, the assumption that X is a child of Z implies that there can be no path between Z and Y, which is active given $\boldsymbol{X}$, i.e. we must have

$$Z \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X} \ . \tag{5}$$

But from $X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X}$ and $X \not\perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X} \cup \{Z\}$, Lemma 13 allows us to conclude that $Z \not\perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X}$, which contradicts (5). □

We thus have that the two independence relationships $X \perp\!\!\!\perp_P Y \mid \boldsymbol{X}$ and $X \not\perp\!\!\!\perp_P Y \mid \boldsymbol{X} \cup \{Z\}$, which holds for each $Z \in \boldsymbol{E}_P^{Y|X}$ and in any distribution similar to P on X, imply graphical consequences. Formalising the connection, we have a criterion for identifying some locally compelled arcs:

Lemma 15. *Let P be a DAG faithful probability distribution over $\boldsymbol{V}$, $X \in \boldsymbol{V}$, $Y \in \boldsymbol{S}_P^X$, and $\boldsymbol{M}$ a maximal separating set of Y from X wrt P. Then for any perfect map $\mathcal{G}$ of P, each member Z of $\mathrm{mb}_P(X) \setminus (\boldsymbol{M} \cup \{Y\} \cup \boldsymbol{S}_P^X)$ is a locally compelled child of X. That is, Z is in $\mathrm{ch}_{\mathcal{G}}^*(X)$.*

Proof. We need to show that Z is a child of X in any perfect map of any probability distribution P_2 similar to P on $\{X\}$. Since $\boldsymbol{M}$ is a maximal separating set, it follows that $Z \in \boldsymbol{E}_P^{Y|X}$. Since P and P_2 are similar on $\{X\}$, $Y \in \boldsymbol{S}_P^X$, and $Z \in \boldsymbol{E}_P^{Y|X} \setminus \boldsymbol{S}_P^X$, we have that $Y \in \boldsymbol{S}_{P_2}^X$ and $Z \in \boldsymbol{E}_{P_2}^{Y|X} \setminus \boldsymbol{S}_{P_2}^X$. Hence, there is a (maximal separating) set $\boldsymbol{M}_2 \subseteq \mathrm{mb}_{P_2}(X)$ not including Z, such that $X \perp\!\!\!\perp_{P_2} Y \mid \boldsymbol{M}_2$ and $X \not\perp\!\!\!\perp_{P_2} Y \mid \boldsymbol{M}_2 \cup \{Z\}$. The result now follows from Corollary 11 and Lemma 14. □

Specifically, we have that arcs participating in a v-structure must be locally compelled:

Lemma 16. *Let P be a DAG faithful probability distribution over $\boldsymbol{V}$, $X, Y, Z \in \boldsymbol{V}$, and $\mathcal{G}$ be a perfect map of P. If (X, Z, Y) is a v-structure in $\mathcal{G}$, then $Y \in \boldsymbol{S}_P^X$, $Z \in \boldsymbol{E}_P^{Y|X} \setminus \boldsymbol{S}_P^X$, and $X \to Z$ and $Y \to Z$ are both locally compelled.*

Proof. We only show that $X \to Z$ is locally compelled, as the case of $Y \to Z$ is similar.

Since Y is not adjacent to X, Corollary 11 implies that $Y \in \boldsymbol{S}_P^X$, and hence that $\boldsymbol{E}_P^{Y|X}$ is well-defined. Clearly, Z cannot be in any maximal separating set of Y from X wrt P, as $\mathcal{G}$ is a perfect map of P and $X \not\perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X} \cup \{Z\}$ for all sets $\boldsymbol{X}$. Furthermore, as Z is adjacent to X, Corollary 11 ensures that $Z \notin \boldsymbol{S}_P^X$. The result now follows from Lemma 15. □

From Corollary 7, Definition 12, and Lemma 16, we thus have the needed result:

Corollary 17. *For any DAG $\mathcal{G}$ and any arc participating in a v-structure, we have that this arc is locally compelled in $\mathcal{G}$. Furthermore, we have that each locally compelled arc in $\mathcal{G}$ is compelled in $\mathcal{G}$.*

To see that each of the inclusions are proper, consider the graph in Figure 6. Here the arcs $A \to D$, $A \to C$, and $B \to C$ are all locally compelled, but only the latter two participate in v-structures. Furthermore, consider the graph in Figure 9. Here each arc is compelled, but the arc $C \to D$ is not locally compelled. That is, a probability distribution having as its perfect map the graph resulting from reversing this arc, would still be similar on C. In this resulting graph, however, the arc $D \to C$ *would* be locally compelled.

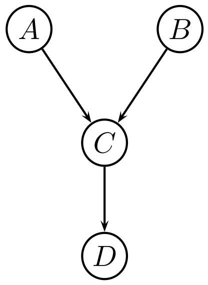


Figure 9: *$Z \to V$ is compelled but not locally compelled.*

At this point, we have that it is sufficient to require of an algorithm uncovering $\mathcal{G}_P^*$, for some DAG faithful distribution P, that it identifies the skeleton of $\mathcal{G}_P^*$, identifies all locally compelled arcs, directs them, and then exchanges arcs not participating in v-structures with links. Moreover, any member of $\mathcal{G}_P$'s equivalence class can be constructed by identifying all locally compelled arcs and then directing the remaining arcs without introducing cycles or new v-structures. Formally, we can restate Corollary 17 in terms that emphasise the connection between similarity and equivalence:

Corollary 18. *Let $\mathcal{G}_1 = (\boldsymbol{V}, \boldsymbol{E}_1)$ and $\mathcal{G}_2 = (\boldsymbol{V}, \boldsymbol{E}_2)$ be perfect maps of distributions P_1 and P_2. Then P_1 is similar to P_2 on $\boldsymbol{V}$ iff $\mathcal{G}_1$ is equivalent to $\mathcal{G}_2$.*

5.2 Soundness and Completeness

Now we proceed by asserting that the arcs directed by our method are indeed locally compelled, and that no locally compelled arcs are left undirected. The main difficulty with applying Lemma 15 to guarantee that LEARNFRAGMENT$(\cdot)$ only directs locally compelled arcs, is that it does not check that the separating set $\boldsymbol{Z}$ found in Line 8 is maximal, so we cannot conclude that any found Z must be in $\boldsymbol{E}_P^{Y|X}$. We show that this is not a problem, as even if $\boldsymbol{Z}$ is not maximal, there must be some other variable W such that $Z \in \boldsymbol{E}_P^{W|X}$, and hence that Z is a locally compelled child:

Lemma 19. *Let P be a DAG faithful probability distribution over $\boldsymbol{V}$, $X, Y \in \boldsymbol{V}$, $\boldsymbol{X} \subseteq \mathrm{mb}_P(X) \setminus \{Y\}$, and $Z \in \mathrm{mb}_P(X) \setminus (\boldsymbol{X} \cup \boldsymbol{S}_P^X \cup \{Y\})$. If $X \perp\!\!\!\perp_P Y \mid \boldsymbol{X}$ and $X \not\perp\!\!\!\perp_P Y \mid \boldsymbol{X} \cup \{Z\}$, then there is $W \in \boldsymbol{S}_P^X$ such that $Z \in \boldsymbol{E}_P^{W|X}$.*

Proof. Let $\mathcal{G}$ be a perfect map of P. We show that there is a $W \in \boldsymbol{S}_P^X$ and $\boldsymbol{X}' \subseteq \mathrm{mb}_\mathcal{G}(X) \setminus \{W, Z\}$ such that

$$X \perp\!\!\!\perp_\mathcal{G} W \mid \boldsymbol{X}' \tag{6}$$

and

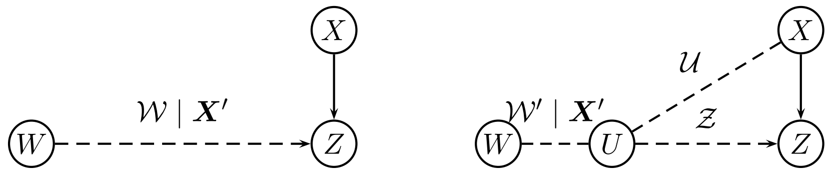
$$X \not\perp\!\!\!\perp_\mathcal{G} W \mid \boldsymbol{X}' \cup \boldsymbol{X}'' \cup \{Z\} \,, \tag{7}$$

for all $\boldsymbol{X}'' \subseteq \mathrm{mb}_{\mathcal{G}}(X) \setminus (\boldsymbol{X}' \cup \{W, Z\})$. $\mathcal{G}$ being a perfect map of P then implies that there is some maximal separating set $\boldsymbol{M}$ of W from X wrt P, where $\boldsymbol{X}' \subseteq \boldsymbol{M} \subseteq \boldsymbol{X}' \cup \boldsymbol{X}''$, and that $Z \notin \boldsymbol{M}$. That is, $Z \in \boldsymbol{E}^{W|X}$, which will prove the lemma.

From the assumptions, it is clear that there is at least one $W \in \boldsymbol{S}_P^X$ (namely Y), such that (6) and

$$X \not\perp\!\!\!\perp_{\mathcal{G}} W \mid \boldsymbol{X}' \cup \{Z\} \,, \tag{8}$$

if we choose $\boldsymbol{X}'$ to be $\boldsymbol{X}$. From Corollary 11 and Lemma 14 it follows that Z is a child of X in $\mathcal{G}$. From Lemma 13 we get that there must be at least one active path $\mathcal{W}$ between W and Z given $\boldsymbol{X}'$. WLOG let W be chosen such that $\mathcal{W}$ is as short as possible. We prove that the existence of $\mathcal{W}$ implies that (7) is fulfilled.



(a) (b)

Figure 10: *Finding a node U to break the connection to W.*

First note that the last arc in $\mathcal{W}$ (connecting to Z — see Figure 10(a)) must be directed into Z, as otherwise there would be an active path from X to W given $\boldsymbol{X}'$, contradicting (6). Also note that if the length of $\mathcal{W}$ is 1, then we have the situation where Z is a child of both W and X, and hence that $\mathcal{W}$ trivially ensures that (7) is fulfilled. Assume therefore that $\mathcal{W}$ has length 2 or more.

Here, we shall prove by contradiction that the existence of $\mathcal{W}$ implies (7): That is, we assume that there is a set $\boldsymbol{X}'' \subseteq \mathrm{mb}_{\mathcal{G}}(X) \setminus (\boldsymbol{X}' \cup \{W, Z\})$, such that

$$X \perp\!\!\!\perp_{\mathcal{G}} W \mid \boldsymbol{X}' \cup \boldsymbol{X}'' \cup \{Z\} \,, \tag{9}$$

and prove that this means that W cannot have been chosen such that $\mathcal{W}$ is as short as possible. As $\mathcal{W}$ is active given $\boldsymbol{X}'$ and Z is a child of X, it follows that the nodes in $\boldsymbol{X}''$ must render $\mathcal{W}$ inactive, which can only happen if some of the nodes in the chosen $\boldsymbol{X}''$ lie on $\mathcal{W}$. Let U be the node in $\boldsymbol{X}''$ closest to W on $\mathcal{W}$, such that the connection at U is either serial or diverging (meaning that U renders $\mathcal{W}$ inactive) (see Figure 10(b)). There are three possible scenarios:

1. $\mathcal{W} : (W \cdots \leftarrow U \rightarrow \cdots \rightarrow Z)$
2. $\mathcal{W} : (W \cdots \leftarrow U \leftarrow \cdots \rightarrow Z)$
3. $\mathcal{W} : (W \cdots \rightarrow U \rightarrow \cdots \rightarrow Z)$

Let $\mathcal{W}'$ be the part of $\mathcal{W}$ running between W and U and $\mathcal{Z}$ be the part between U and Z. We show that U can serve the same role as W, and hence obtain the contradiction. Assume first that either Scenario 1 or 2 are the case: If $X \not\perp\!\!\!\perp_{\mathcal{G}} U \mid \boldsymbol{X}'$, then there is an active path $\mathcal{U}$ between X and U given $\boldsymbol{X}'$, and hence also an active path from W to X given $\boldsymbol{X}'$ formed by joining $\mathcal{W}'$ and $\mathcal{U}$. But that means that $X \not\perp\!\!\!\perp_{\mathcal{G}} W \mid \boldsymbol{X}'$, which contradicts (6). Thus, we must have that $X \perp\!\!\!\perp_{\mathcal{G}} U \mid \boldsymbol{X}'$. Furthermore, since U is on $\mathcal{W}$, it follows that $X \not\perp\!\!\!\perp_{\mathcal{G}} U \mid \boldsymbol{X}' \cup \{Z\}$. But then (6) and (8) is satisfied by choosing W as U, by the virtue of the active path $\mathcal{Z}$, and hence $\mathcal{W}$ is not the shortest such path. Thus none of Scenarios 1 and 2 can be the case.

Assume then that Scenario 3 is the case: First we note that it cannot be the case that $X \perp\!\!\!\perp_{\mathcal{G}} U \mid \boldsymbol{X}'$ since $\mathcal{Z}$ is active given $\boldsymbol{X}'$ and this implies that (6) and (8) would be satisfied by choosing W as U, thus violating the definition of W as being closest to Z. Hence, $X \not\perp\!\!\!\perp_{\mathcal{G}} U \mid \boldsymbol{X}'$ meaning that there is a path $\mathcal{U}$ from X to U active given $\boldsymbol{X}'$. Since $\mathcal{W}'$ is also active given $\boldsymbol{X}'$ and (6) is assumed, it follows that the arc attached to U in $\mathcal{U}$, must be an arc going into U. But since that results in a converging connection at U joining $\mathcal{W}'$ and $\mathcal{U}$, and $X \perp\!\!\!\perp_{\mathcal{G}} W \mid \boldsymbol{X}'$, we have that no descendant of U can be in $\boldsymbol{X}'$. Specifically, we have that $\mathcal{Z}$ cannot contain any nodes of $\boldsymbol{X}'$ or nodes with a descendant in $\boldsymbol{X}'$, and hence that it must be a directed path from U to Z.

Assume that $\mathcal{Z}$ contains some nodes that are adjacent to X. None of these can be parents of X, as the arc from the parent node to X along with the active path $\mathcal{W}$, would mean that $X \not\perp\!\!\!\perp_{\mathcal{G}} W \mid \boldsymbol{X}'$, contradicting (6). Thus nodes on $\mathcal{Z}$ in $\mathrm{adj}_{\mathcal{G}}(X)$ are children of X. U itself must be nonadjacent to X, as U cannot be a child of X: That would imply that $X \not\perp\!\!\!\perp_{\mathcal{G}} U \mid \boldsymbol{X}' \cup \boldsymbol{X}'' \cup \{Z\}$. Moreover $\mathcal{W}'$ is active given $\boldsymbol{X}' \cup \boldsymbol{X}'' \cup \{Z\}$, would form a converging connection with the arc from X to U at U. As $U \in \boldsymbol{X}''$ that would mean $X \not\perp\!\!\!\perp_{\mathcal{G}} W \mid \boldsymbol{X}' \cup \boldsymbol{X}'' \cup \{Z\}$, which violates assumption (9). Hence, U must be nonadjacent to X.

Since U is not adjacent to X, we thus have that there is some node on $\mathcal{Z}$ nonadjacent to X and closer to Z on $\mathcal{Z}$ than any other such node. Let V be this node. Since Z is a child of X and a direct descendant of V, it is easy to see that both (6) and (8) are satisfied by choosing W as V and thus that W was not chosen as close to Z as possible after all. This final contradiction proves the lemma. $\square$

Combining Lemmas 15 and 19 we thus have a guarantee that the arcs directed by LearnFragment($\cdot$) are locally compelled:

Corollary 20. *Let P be a DAG faithful probability distribution over $\boldsymbol{V}$, $X \in \boldsymbol{V}$, $Y \in \boldsymbol{S}_P^X$, $\boldsymbol{X} \subseteq \mathrm{mb}_P(X) \setminus \{Y\}$, and $Z \in \mathrm{mb}_P(X) \setminus (\boldsymbol{X} \cup \boldsymbol{S}_P^X \cup \{Y\})$. If $X \perp\!\!\!\perp_P Y \mid \boldsymbol{X}$ and $X \not\perp\!\!\!\perp_P Y \mid \boldsymbol{X} \cup \{Z\}$, then for any DAG $\mathcal{G}$, which is a perfect map of P, Z is in $\mathrm{ch}^*_{\mathcal{G}}(X)$.*

Following the reasoning in the proof of Lemma 16, it is also clear that an arc $X \rightarrow Y$ participating in a v-structure (X, Y, Z) will be found by LearnFragment($\cdot$), and hence we have that LearnFragment($\cdot$) would uncover a perfect map of a distribution if it was invoked for all nodes. Thus, LearnFragment($\cdot$) can be said to be complete.

However, in general, some nodes will have their graph fragment constructed by ExtractFragment($\cdot$). For these nodes, we are directing arcs according to their direction in the current network. To see that it is not sufficient to simply direct the arcs participating in v-structures, consider two distributions with perfect maps as shown in Figures 11(a) and 11(b). The distributions are similar on A and LearnFragment($\cdot$) only ever directs arcs out of the node it is invoked on, so the arc from A to C in (b) must be identified by ExtractFragment($\cdot$). This will not happen if only arcs participating in v-structures in (a) are identified, and we therefore direct more arcs than simply those of v-structures. Thus, we need more elaborate arguments to show that ExtractFragment($\cdot$) is correct, in that we need guarantees both that the arcs directed by it are all locally compelled (soundness), and that no locally compelled arc is left undirected (completeness).

We first show that ExtractFragment($\cdot$) is sound wrt the current graph, since each arc it identifies implies that the conditions for Lemma 15 holds for any distribution with the current graph as its perfect map, and hence that LearnFragment($\cdot$) would have learned this arc as well if run with an oracle based on such a distribution.

Lemma 21. *Let P be a DAG faithful probability distribution over $\boldsymbol{V}$, $X \in \boldsymbol{V}$, and $\mathcal{G}$ be a perfect map of P. The following are equivalent statements:*

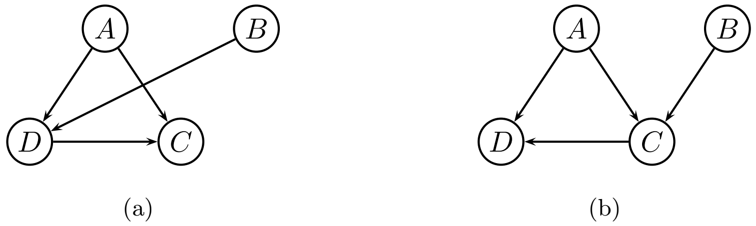


Figure 11: *(a) and (b) are perfect maps of distributions similar on X.*

1. *$Z \in \mathrm{ch}_{\mathcal{G}}(X)$ and there is a $W \in \mathrm{mb}_{\mathcal{G}}(X) \setminus \mathrm{adj}_{\mathcal{G}}(X)$ such that Z is a descendant of W through a path encompassing only children of X,*

2. *$Z \in \mathrm{ch}_{\mathcal{G}}(X)$ and there is a v-structure (X, Z', Y) in $\mathcal{G}$ such that $Z = Z'$ or Z is a descendant of Z', and*

3. *there is $Y \in \boldsymbol{S}_P^X$ such that $Z \in \boldsymbol{E}_P^{Y|X} \setminus \boldsymbol{S}_P^X$.*

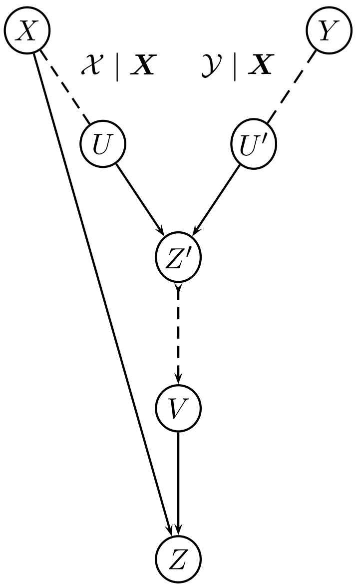


Figure 12: *A construction used to identify ancestors of Z nonadjacent to X.*

Proof. That 1 implies 2 is obvious if Y is chosen to be W. That 2 implies 3, is an immediate consequence of the definition of d-separation, since Z cannot be in any separating set of X and Y, and the path from Z' to Z cannot be blocked without the blocking node itself opening a connection between X and Y. So we only show that 3 implies 1: Since $Z \in \boldsymbol{E}_P^{Y|X} \setminus \boldsymbol{S}_P^X$ we have from Lemma 15 that Z is a child of X in $\mathcal{G}$. Moreover, we know that there is a set $\boldsymbol{X} \subseteq \mathrm{mb}_P(X)$ such that $X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X}$, but $X \not\perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X} \cup \{Z\}$, and thus by Lemma 13 that in $\mathcal{G}$ there is a node Z' (see Figure 12) with nonadjacent parents U and U', such that Z is a descendant of Z' on path $\mathcal{Z}$, and there are active paths $\mathcal{X}$ and $\mathcal{Y}$ given $\boldsymbol{X}$ between X and U and between U' and Y. Let V be the parent of Z on $\mathcal{Z}$, which implies that $V \in \mathrm{mb}_{\mathcal{G}}(X)$. If V is furthermore not in $\mathrm{adj}_{\mathcal{G}}(X)$ then we have found the node W that we are looking for, so assume that this is not the case. As V is a descendant of Z' it follows that it is also in $\boldsymbol{E}_P^{Y|X}$. Then Lemma 15 can be applied again with V taking the place of Z, and we get that V is a child of X. Iterating in this manner we either find the W, we are looking for, or we get that each node on $\mathcal{Z}$ must be a child of X. Specifically this holds for Z'. But then either U' is the node W we are looking for, or there is an active path from X to Y given $\boldsymbol{X}$, consisting of the connection between X and U' along with $\mathcal{Y}$. The latter possibility is precluded by the assumption $X \perp\!\!\!\perp_{\mathcal{G}} Y \mid \boldsymbol{X}$. □

As a direct consequence of this graphical result and d-separation properties on graphs, we have that LearnFragments($\cdot$) could not find any more locally compelled arcs, even if it considered the entire set of variables, rather than simply the variables in the Markov boundary, when finding variables that are separable:

Corollary 22. *Let P be a DAG faithful probability distribution over $\boldsymbol{V}$, $X, Z \in \boldsymbol{V}$, and $\mathcal{G}$ be a perfect map of P. There is $Y \in \boldsymbol{S}_P^X$ such that $Z \in \boldsymbol{E}_P^{W|X} \setminus \boldsymbol{S}_P^X$ iff there is $W \in \boldsymbol{S}_P^X \cap \mathrm{mb}_P(X)$ such that $Z \in \boldsymbol{E}_P^{W|X} \setminus \boldsymbol{S}_P^X$.*

A consequence of this corollary is that ExtractFragment($\cdot$), when run on a graph $\mathcal{G}$, finds all the locally compelled arcs that would have been found by LearnFragment($\cdot$) if it was run with an oracle based on a probability distribution with $\mathcal{G}$ as a perfect map. In a sense, the two methods are equivalent, but base their decisions on different sources.

Finally, we have the completeness result, where we show that all the locally compelled arcs in a graph would be identified by ExtractFragment($\cdot$).

Lemma 23. *Let P be a DAG faithful probability distribution over $\boldsymbol{V}$, $X, Z \in \boldsymbol{V}$, and $\mathcal{G}$ be a perfect map of P. We then have that $Z \in \mathrm{ch}^*_{\mathcal{G}}(X)$ implies that there is a $W \in \mathrm{mb}_{\mathcal{G}}(X) \setminus \mathrm{adj}_{\mathcal{G}}(X)$ such that Z is a descendant of W through a path encompassing only children of X.*

Proof. We prove the lemma by contradiction. That is, we assume that there is no $W \in \mathrm{mb}_{\mathcal{G}}(X) \setminus \mathrm{adj}_{\mathcal{G}}(X)$, such that Z is a descendant of W through a path encompassing only children of X, and show that this means that there is a probability distribution similar to P on $\{X\}$ but having $X \leftarrow Z$ instead of $X \rightarrow Z$. This contradicts that $X \rightarrow Z$ is a locally compelled arc.

First note that the lack of a suitable W implies that $X \rightarrow Z$ cannot participate in any v-structures in $\mathcal{G}$. Next, construct the graph $\mathcal{G}_2$ which is identical to $\mathcal{G}$ except for the direction of the arc between X and Z. We show that any probability distribution P_2 having $\mathcal{G}_2$ as a perfect map is similar to P on X.

We need to show that all three bullets of Definition 6 are satisfied. First, as $X \rightarrow Z$ is not participating in any v-structures, and this is the only arc that is different in $\mathcal{G}_2$, we have that $\mathrm{mb}_{\mathcal{G}_2}(X) = \mathrm{mb}_{\mathcal{G}}(X)$, so Bullet 1 is satisfied. Similarly, the fact that no adjacencies are changed implies by Corollary 11 that Bullet 2 is satisfied. So all that is left to prove is that Bullet 3 is satisfied. Assume this is not the case. Then there is some node Y such that $\boldsymbol{E}_P^{Y|X} \setminus \boldsymbol{S}_P^X \neq \boldsymbol{E}_{P_2}^{Y|X} \setminus \boldsymbol{S}_{P_2}^X = \boldsymbol{E}_{P_2}^{Y|X} \setminus \boldsymbol{S}_P^X$. Let Z' be a node having either left or entered such an enabling set due to the arc reversal. First note that the assumption that Z is in no such sets means that if $Z' = Z$, then Z must have entered such a set. But then Lemma 15 states that $X \rightarrow Z$ is a (locally compelled) arc in $\mathcal{G}_2$ which contradicts the definition of $\mathcal{G}_2$. Thus $Z' \neq Z$. We prove this is impossible, too, reaching the required contradiction.

First assume that $Z' \in \boldsymbol{E}_P^{Y|X} \setminus \boldsymbol{E}_{P_2}^{Y|X}$. According to Lemma 21 this means that in $\mathcal{G}$ there Z' is a child of X and that there is a path $\mathcal{Y}$ from Y to Z' running through only children of X, and that this is not the case in $\mathcal{G}_2$. But $\mathcal{Y}$ cannot be destroyed by reversing $X \rightarrow Z$ only, as that would require the arc to be on $\mathcal{Y}$, which contradicts the definition of $\mathcal{Y}$, so we must have that in $\mathcal{G}_2$ X is no longer a parent of Z'. But then $Z' = Z$ which is impossible.

Finally, assume that $Z' \in \boldsymbol{E}_{P_2}^{Y|X} \setminus \boldsymbol{E}_{P}^{Y|X}$. Again, we have from Lemma 21 that Z' is a child of X and that there is a path $\mathcal{Y}$ from Y to Z' running through only children of X, but this time in $\mathcal{G}_2$ only. Again, we have that $\mathcal{Y}$ cannot be destroyed by reversing $X \leftarrow Z$ only, as that would require the arc to be on $\mathcal{Y}$, so we must have that in $\mathcal{G}$ X is no longer a parent of Z'. But this is impossible. □

5.3 Correctness Proof

Summing up on the results above, we have:

Corollary 24. *Let P be a DAG faithful probability distribution over $\boldsymbol{V}$, $\mathcal{G}$ a perfect map of P, and $X, Z \in \boldsymbol{Z}$. Then the following are equivalent statements:*

1. *$X \to Z$ is a locally compelled arc in $\mathcal{G}$,*
2. *there is $Y \in \boldsymbol{S}_P^X$ such that $Z \in \boldsymbol{E}_P^{Y|X} \setminus \boldsymbol{S}_P^X$,*
3. *there is $Y \in \boldsymbol{S}_P^X \cap \mathrm{mb}_P(X)$ such that $Z \in \boldsymbol{E}_P^{Y|X} \setminus \boldsymbol{S}_P^X$,*
4. *$Z \notin \boldsymbol{S}_P^X$, and there is $Y \in \mathrm{mb}_P(X)$ and $\boldsymbol{X} \subseteq \mathrm{mb}_P(X)$ such that $X \perp\!\!\!\perp_P Y \mid \boldsymbol{X}$ and $X \not\perp\!\!\!\perp_P Y \mid \boldsymbol{X} \cup \{Z\}$,*
5. *$Z \in \mathrm{ch}_{\mathcal{G}}(X)$, and there is $Y \in \mathrm{mb}_{\mathcal{G}}(X) \setminus \mathrm{adj}_{\mathcal{G}}(X)$ such that there is a directed path from Y to Z in $\mathcal{G}$ running through only nodes in $\mathrm{ch}_{\mathcal{G}}(X)$, and*
6. *$Z \in \mathrm{ch}_{\mathcal{G}}(X)$, and there is a v-structure (X, Z', Y) in $\mathcal{G}$ such that $Z = Z'$ or Z is a descendant of Z'.*

Proof. That 2 implies 1, follows from Lemma 15. We have 2 iff 3 from Corollary 22. That 4 implies 2, follows from Lemma 19. That 3 implies 4, follows from the definitions of $\boldsymbol{S}_P^X$ and $\boldsymbol{E}_P^{Y|X}$. Lemma 21 yields that 2 iff 5 iff 6. Finally, that 1 implies 5 follows from Lemma 23. □

Thus, there are strong correlations between the abstract probabilistic definition of locally compelled, the conditional independence oriented definition of enabling sets, and strictly graphical notions. All of this allow us to finally state the proof of Theorem 8:

Theorem 8. *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. *$r > 2k$, 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*
4. UncoverMarkovBoundary*$(X,\, \mathcal{D}')$ returns $\mathrm{mb}_{P_2}(X)$ if $\mathcal{D}'$ is sampled from P_2 only, and*
5. *the oracle used in* LearnFragment*$(\cdot)$ is correct,*

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

Proof. Given Assumptions 2 and 3 it is obvious that ADAPT$(\cdot)$ does nothing except exactly once, when it calls UPDATENET$(\cdot)$ on $\mathcal{B}_1$, $\boldsymbol{V} \setminus \boldsymbol{I}$, and the first k cases $\mathcal{D}'$ of the partition of $\mathcal{D}$ sampled from P_2.

According to Theorem 3, if we can prove that

(i) two nodes are adjacent in $\mathcal{G}_2$ iff they are adjacent in $\mathcal{G}_{P_2}$, and

(ii) an arc participating in a v-structure in $\mathcal{G}_2$ is in $\mathcal{G}_{P_2}$ and vice-versa,

then we have that $\mathcal{G}_2^*$ is identical to $\mathcal{G}_{P_2}^*$. The result then follows from the correctness of Rules 1 to 3b on page 14, which has been proved in (Spirtes et al. 2001).

i) Let $X, Y \in \boldsymbol{V}$ be adjacent in $\mathcal{G}_2$. It is either the case that they were set so by LEARNFRAGMENT$(\cdot)$ or EXTRACTFRAGMENT$(\cdot)$. In the former case, this means that LEARNFRAGMENT$(\cdot)$ was not able to find a set $\boldsymbol{Z} \in \boldsymbol{R}$ such that $I_{\mathcal{D}'}(X, Y|\boldsymbol{Z})$. By Assumptions 4 and 5, this implies that there is no $\boldsymbol{Z} \in \mathrm{mb}_{P_2}(X)$ such that $X \perp\!\!\!\perp_{P_2} Y \mid \boldsymbol{Z}$, and hence that $Y \notin \boldsymbol{S}_{P_2}^X$, which by Corollary 11 means that X and Y are adjacent in $\mathcal{G}_{P_2}$. Assume then that X and Y were set adjacent by EXTRACTFRAGMENT$(\cdot)$. This means that X and Y are adjacent in $\mathcal{G}_1$, and by Assumption 3, that at least one of the nodes, say X, is in $\boldsymbol{I}$. We have from Assumption 1 and Corollary 11 that $Y \notin \boldsymbol{S}_{P_1}^X$, and as $X \in \boldsymbol{I}$, it follows from Definition 6 that $Y \notin \boldsymbol{S}_{P_2}^X$. One final application of Corollary 11 yields that X and Y are adjacent in $\mathcal{G}_{P_2}$.

Next, assume that X and Y are nonadjacent in $\mathcal{G}_2$. Assume first that at least one of the nodes, say X, is not in $\boldsymbol{I}$. LEARNFRAGMENT$(\cdot)$ leaves X and Y nonadjacent, only if it finds $\boldsymbol{Z} \subseteq \boldsymbol{R}$ such that $I_{\mathcal{D}'}(X, Y|\boldsymbol{Z})$. By Assumption 5 this means that $X \perp\!\!\!\perp_{P_2} Y \mid \boldsymbol{Z}$, which by Corollary 11 implies that X and Y are nonadjacent in $\mathcal{G}_{P_2}$. Assume next that both X and Y are in $\boldsymbol{I}$. Then EXTRACTFRAGMENT$(\cdot)$ have neglected to add an adjacency between the two because there is none in $\mathcal{G}_1$. From Assumption 1 we can conclude that X and Y are nonadjacent in $\mathcal{G}_{P_1}$, and by Corollary 11 that $Y \in \boldsymbol{S}_{P_1}^X$. Since X is in $\boldsymbol{I}$, Assumption 3 and Definition 6 yield that $Y \in \boldsymbol{S}_{P_2}^X$, and Corollary 11 then implies that X and Y are nonadjacent in $\mathcal{G}_{P_2}$.

ii) Consider an arc $X \to Z$ participating in a v-structure (X, Z, Y) in $\mathcal{G}_{P_2}$. Assume first that $X \notin \boldsymbol{I}$. From the existence of (X, Z, Y), Lemma 16 and Corollary 24 guarantee that there is at least one set $\boldsymbol{Z} \subseteq \mathrm{mb}_{P_2}(X)$ such that $X \perp\!\!\!\perp_{P_2} W \mid \boldsymbol{Z}$ and $X \not\perp\!\!\!\perp_{P_2} W \mid \boldsymbol{Z} \cup \{Z\}$ for some $W \in \mathrm{mb}_{P_2}(X)$. From Assumption 4 we have that $\{X, W\} \cup \boldsymbol{Z}$ is guaranteed to be a subset of $\boldsymbol{R}$ when LEARNFRAGMENT$(\cdot)$ investigates X, and by Assumption 5 this means that the tests $I_{\mathcal{D}'}(X, W|\boldsymbol{Z})$ and $\neg I_{\mathcal{D}'}(X, W|\boldsymbol{Z} \cup \{Z\})$ will succeed. Hence, $\mathcal{G}_2$ will include the arc $X \to Z$.

Assume next that $X \in \boldsymbol{I}$, and assume that $\mathcal{G}_2$ does not include $X \to Z$. We show that this leads to a contradiction. If $\mathcal{G}_2$ does not include an arc from X to Z, this means that Z cannot both be in $\mathrm{ch}_{\mathcal{G}_1}(X)$ and be a descendant of some Z' such that (X, Z', W) is a v-structure in $\mathcal{G}_1$, as that would have caused EXTRACTFRAGMENT$(\cdot)$ to put it there. By Assumption 1 and Corollary 24, this means that there is no $W \in \boldsymbol{S}_{P_1}^X$ such that $Z \in \boldsymbol{E}_{P_1}^{W|X}$. By Assumption 3 and Definition 6 this means that there is no $W \in \boldsymbol{S}_{P_2}^X$ such that $Z \in \boldsymbol{E}_{P_2}^{W|X}$. But according to Corollary 24 and Lemma 16 this is a contradiction with the existence of (X, Z, Y) in $\mathcal{G}_{P_2}$, as Y fulfills the role of W. Hence Z must be a child of X in $\mathcal{G}_2$ after all.

Next, consider an arc $X \to Z$ participating in a v-structure (X, Z, Y) in $\mathcal{G}_2$, and assume first that $X \notin \boldsymbol{I}$. This means that the arc was constructed by LEARNFRAGMENT$(\cdot)$ when X was investigated, because a set $\boldsymbol{Z}$ and node W was found, where $W \in \mathrm{mb}_{P_2}(X)$ by Assumption 4, $I_{\mathcal{D}'}(X, W|\boldsymbol{Z})$, and $\neg I_{\mathcal{D}'}(X, W|\boldsymbol{Z} \cup \{Z\})$. By Assumption 5 this means that we have $X \perp\!\!\!\perp_{P_2} W \mid \boldsymbol{Z}$ and $X \perp\!\!\!\perp_{P_2} W \mid \boldsymbol{Z} \cup \{Z\}$. By Corollary 24 this implies that $X \to Z$ is in $\mathcal{G}_{P_2}$.

Finally, assume that $X \in \boldsymbol{I}$. $X \to Z$ was then constructed by ExtractFragment$(\cdot)$, which means that it is in $\mathcal{G}_1$ and that in $\mathcal{G}_1$ there is a $W \notin \mathrm{adj}_{\mathcal{G}_1}(X)$, such that Z is a descendant of W through a path consisting only of children of X. By Assumption 1 and Corollary 24, this means that $Z \in \boldsymbol{E}_{P_1}^{W|X} \setminus \boldsymbol{S}_{P_1}^{X}$. By Assumption 3 and Definition 6 we have that $W \in \boldsymbol{S}_{P_2}^{X}$ and $Z \in \boldsymbol{E}_{P_2}^{W|X} \setminus \boldsymbol{S}_{P_2}^{X}$, which by Corollary 24 implies that $X \to Z$ is a (locally compelled) arc in $\mathcal{G}_{P_2}$. $\square$

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,

2. if the strict confinement of the algorithm to learning only graph fragments for some nodes chosen by the heuristic cripples it in long term performance, and

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

6.1 Configuration of Algorithm

In the implementation we have used a slightly different algorithm for LearnFragment$(\cdot)$, shown in Algorithm 7, rather than the one presented back in Algorithm 3. Under the assumptions of Theorem 8 the two algorithms should yield equivalent results, but the one in Algorithm 7 should be more robust in the face of flawed independence tests, if the generating distribution is indeed DAG faithful, as the tests used for uncovering adjacent nodes in the alternative algorithm involves smaller conditioning sets. The two algorithms differ in that Lines 4 to 17 in Algorithm 3 has been substituted by Lines 4 to 16 in Algorithm 7. The difference is that here nodes not adjacent to X in $\mathcal{B}_{\mathcal{D}'}$ are uncovered using UncoverNonAdjacents$(X, \boldsymbol{R}, \mathcal{D}')$ prior to (as opposed to during) establishing enabling nodes, which allows us to use the AlgorithmPCD$(\cdot)$ method in (Peña et al. 2005) (restricted to the variables in $\boldsymbol{R}$) to first find variables adjacent to X and then returning the rest as nonadjacent nodes, rather than trying all kinds of separating sets as is done in Algorithm 3. The AlgorithmPCD$(\cdot)$ method uses a greedy approach to establishing adjacent variables, and it is less reliant on large data sets than the approach based on enumerating possible separating sets (Peña et al. 2005).

This approach to establishing nonadjacent nodes means that we have to search for separating sets afterwards for the purpose of finding enabling nodes and hence locally compelled arcs. However, if we fail to find a separating set here, only the direction of arcs in the fragment might be wrong, and not the skeleton in itself, which reduces the negative impact of flawed tests. The AlgorithmPCD$(\cdot)$ method needs an independence oracle, and here we have used the same χ^2 tests that we have used for the tests in MarkovBoundary$(\cdot)$ and for the independence tests in the rest of the 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 $P_{\mathcal{B}}(X_i = x_i)$

Algorithm 7 *Learns a graph fragment for a variable X consistent with $\mathcal{B}^{**}_{\mathcal{D}'}$ and the fragments in $\boldsymbol{G}$.*

procedure LearnFragment(X, $\mathcal{D}'$, $\boldsymbol{G}$)
 $(\mathcal{G}_X \equiv (\boldsymbol{V}, \boldsymbol{E}_X), \boldsymbol{N}) \leftarrow$AlignWithOtherFragments($X$, $\boldsymbol{G}$)
 $\boldsymbol{R} \leftarrow \mathrm{adj}_{\mathcal{G}_X}(X) \cup$MarkovBoundary($X$, $\mathcal{D}'$)
 $\boldsymbol{S} \leftarrow \boldsymbol{N} \cup$UncoverNonAdjacents($X$, $\boldsymbol{R}$, $\mathcal{D}'$)
 $\boldsymbol{E}^{|X} \leftarrow \varnothing$
 for $Y \in \boldsymbol{S}$ **do**
  for $\boldsymbol{Z} \subseteq \boldsymbol{R} \setminus \{Y\}$ **do**
   if $I_{\mathcal{D}'}(X, Y|\boldsymbol{Z})$ **then**
    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**
     if $\neg I_{\mathcal{D}'}(X, Y|\boldsymbol{Z} \cup \{Z\})$ **then**
      $\boldsymbol{E}^{|X} \leftarrow \boldsymbol{E}^{|X} \cup \{Z\}$
     end if
    end for
   end if
  end for
 end for
 for $Y \in \boldsymbol{R} \setminus \boldsymbol{S}$ **do**
  $\boldsymbol{E}_X \leftarrow \boldsymbol{E}_X \cup \{(X, Y)\} \setminus \{(Y, X)\}$
  if $Y \notin \boldsymbol{E}^{|X}$ **then**
   $\boldsymbol{E}_X \leftarrow \boldsymbol{E}_X \cup \{(Y, X)\}$
  end if
 end for
 return $(\boldsymbol{V}, \boldsymbol{E}_X)$
end procedure

in the numerator of (4), we used alternative probabilities $P_u(X_i = x_i)$ 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 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.

Finally, 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.

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 (see e.g. (Koster et al. 2001)) 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 four different methods for generating CPTs for DAGs:

- *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 $\boldsymbol{X}$ using the method of Caprile (1999). 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:

- n the number of nodes in each DAG $\mathcal{G}_i$,

- p the percentage of nodes whose graphical bindings are changed in the transition from each $\mathcal{G}_i$ to $\mathcal{G}_{i+1}$,

- w the maximum induced tree width of each DAG $\mathcal{G}_i$,

- m the number of BNs in the sequence, and

- d the method used for generating each $\boldsymbol{\Phi}_i$ (Hard, Medium, Easy, and T-Hard).

Here we shall report on eight sets of parameters, all summarised 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).

We generated piecewise DAG faithful sample sequences by sampling from 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

Table 1: *Experimental settings.*

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

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, which was the same for all experiments based on sample sequences for a given BN sequence. For each of these experiments we measured the deviance and efficiency of the method wrt three distance measures:

- The KL-distance (KL),
- the number of wrong connections (WC), i.e. links in the skeleton of the learned network, which are not in the skeleton of the generating network, plus links in the skeleton of the generating network but not in the skeleton of the learned network, and
- the number of wrong v-structures (WV), i.e. v-structures in the learned network not in the generating network plus v-structures in the generating network not in the learned network.

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 $\textsc{ShiftInStream}(\cdot)$, 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. So we have four versions of our method:

A: $\tau = 27$ and $\alpha = 0.01$,

B: $\tau = 27$ and $\alpha = 0.05$,

C: $\tau = 31$ and $\alpha = 0.01$, and

D: $\tau = 31$ and $\alpha = 0.05$.

For baseline comparisons we implemented the incremental learning methods of (Lam and Bacchus 1994) and (Friedman and Goldszmidt 1997). The framework presented in Lam and Bacchus

(1994) runs a search for a new BN every k'th observation, using a modified MDL score to evaluate networks. In our experiments, we have set the method to use greedy hill-climbing, but any search will do in principle. The score for a candidate network $\mathcal{G}_i$ given a currently maintained network $\mathcal{G}_{i-1}$ and the last k cases of the data stream $\mathcal{D}_k$ is the sum of an optimal encoding of $\mathcal{G}_i$, an optimal encoding of the differences between $\mathcal{G}_i$ and $\mathcal{G}_{i-1}$, an optimal encoding of the probability parameters needed to extend $\mathcal{G}_i$ to a full BN $\mathcal{B}_i$, and an Huffman encoding of $\mathcal{D}_k$ using the probabilities $P_{\mathcal{B}_i}$. The reasoning behind the scoring function is that old observations have an impact on the search for a new network through the encoding of the differences between the new graph and the old one, while new observations influence the search through the size of the Huffman encoding of the data.

The approach of Friedman and Goldszmidt (1997) is based on maintaining a set $\boldsymbol{S}$ of sufficient statistics for a *search boundary* defined by the current network $\mathcal{G}$: A graph $\mathcal{G}'$ is in the search boundary of $\mathcal{G}$, if it is the result of adding, removing, or reversing a single arc in $\mathcal{G}$. The set of sufficient statistics $\boldsymbol{S}$ is the least set such that, for each $\mathcal{G}'$ in the search boundary of $\mathcal{G}$ and $X \in \boldsymbol{V}$, $\boldsymbol{S}$ includes a potential $n^{\boldsymbol{X}}$, such that $X \cup \mathrm{pa}_{\mathcal{G}'}(X) \subseteq \boldsymbol{X}$, and such that for each $n^{\boldsymbol{X}} \in \boldsymbol{S}$ there is some graph $\mathcal{G}'$ in the search boundary, where $\boldsymbol{X} = X \cup \mathrm{pa}_{\mathcal{G}'}(X)$. After reception of every k'th observation, the set $\boldsymbol{S}$ of sufficient statistics for the search boundary, is updated, by first dropping statistics that are no longer needed, and adding new needed ones, and then adding the last k observations $\mathcal{D}_k$ to each of them. Then a search for a new network is run using the current set of sufficient statistics rather than the entire database of all received cases so far. Since sufficient statistics are only maintained for a limited number of sets, there are graph structures for which a score cannot be calculated, so these are assigned the worst possible score during this search. Whenever a search results in a new graph, the set of sufficient statistics is altered by dropping some sufficient statistics and adding others to fit the search boundary of the new graph. As statistics is dropped and added as the learning takes place, each sufficient statistics can be defined over different sets of observations. This is a problem as most scoring functions for BNs assume that this is not the case. Friedman and Goldszmidt (1997) show how the BDe and MDL scoring functions can be augmented to take this fact into account. In our experiments, we used the BDe scoring function with a prior over network structures that penalised models depending on their structural difference to the current model, as described in (Heckerman et al. 1995), and implemented both a version using greedy hill-climbing as the search step and a version using simulated annealing. For both of these search heuristics and the search of the (Lam and Bacchus 1994) method, we limited them to investigating nets with a maximum of four parents for a single node to reduce running time. So the baseline comparison methods are:

E: (Lam and Bacchus 1994) with greedy hill-climbing,

F: (Friedman and Goldszmidt 1997) with greedy hill-climbing, and

G: (Friedman and Goldszmidt 1997) with greedy Simulated annealing.

Both the methods of 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 $2k$ in conflict measure histories ShiftInStream($\cdot$) needs to evaluate. For fairness in comparison we chose to have Methods E to G learn every kth case, 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 10. Each reported number is the mean of the five sample sequences drawn from the listed BN sequence. The best number (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 mix of deadline pressures, instable computers, and long evaluation times for KL-distances, we have neglected to report on the performance of Methods B and D in experiments based on Sequences 5.1 to 8.3.

The results for deviance of the KL-distance in Tables 2 and 3 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. There appears to be a difference between sequences of Settings 2 and 3 and the rest: Apparently, the score-based approach of Methods F and G are better suited to the Easy and Medium settings for CPT generation, whereas the constraint-based approach of Methods A to D is better for the Hard and T-Hard settings. We have no better explanation for this than that the two settings perhaps results in more protrusive local maxima than for the remaining sequences, and that the score-based learners are better at exploiting this.

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 based on Sequence 1.1 in Figures 13 and 14. (To reduce clutter in the plots, we have only shown the performance of Methods A, D, E, and F.) Of course, we cannot make any general statements based on this one experiment (and space restrictions prevent us from showing plots of the remaining 159 experiments), 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 4), 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 gets 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.

Looking at efficiency in Tables 5 and 6, what is most interesting is 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

Table 2: *dev(KL) when starting with correct network.*

	A-100	B-100	C-100	D-100	E-100	F-100	G-100
Sequence 1.1	124.0	**97.4**	109.0	115.0	247.0	199.0	232.0
Sequence 1.2	101.0	81.1	94.2	**74.0**	172.0	101.0	108.0
Sequence 1.3	57.9	52.8	66.3	**38.1**	202.0	126.0	145.0
Sequence 1.4	149.0	*137.0*	188.0	164.0	242.0	147.0	165.0
Sequence 1.5	116.0	119.0	90.0	**84.5**	269.0	172.0	189.0
Sequence 2.1	57.3	60.8	58.2	**42.3**	133.0	80.3	79.5
Sequence 2.2	161.0	153.0	174.0	*125.0*	304.0	139.0	196.0
Sequence 2.3	137.0	151.0	**113.0**	126.0	322.0	208.0	226.0
Sequence 2.4	81.2	75.3	54.4	**53.6**	206.0	95.8	106.0
Sequence 2.5	97.8	87.4	83.3	118.0	167.0	**67.0**	84.5
Sequence 3.1	227.0	215.0	216.0	195.0	366.0	*175.0*	244.0
Sequence 3.2	250.0	277.0	280.0	258.0	587.0	**187.0**	375.0
Sequence 3.3	43.1	**25.9**	47.5	35.1	166.0	34.3	37.5
Sequence 3.4	28.8	26.5	27.0	30.6	90.8	*23.9*	34.7
Sequence 3.5	225.0	203.0	214.0	224.0	406.0	*180.0*	223.0
Sequence 4.1	*105.0*	106.0	119.0	127.0	245.0	112.0	160.0
Sequence 4.2	126.0	134.0	135.0	**114.0**	295.0	181.0	219.0
Sequence 4.3	104.0	103.0	102.0	*85.7*	190.0	102.0	132.0
Sequence 4.4	116.0	83.0	119.0	**72.4**	281.0	176.0	184.0
Sequence 4.5	118.0	104.0	91.0	*88.4*	198.0	107.0	142.0
Sequence 5.1	120.0	N/A	**111.0**	N/A	211.0	1082.0	650.0
Sequence 5.2	**226.0**	N/A	227.0	N/A	275.0	1092.0	1087.0
Sequence 5.3	263.0	N/A	**196.0**	N/A	306.0	1320.0	1127.0
Sequence 6.1	*426.0*	N/A	466.0	N/A	433.0	2081.0	1394.0
Sequence 6.2	421.0	N/A	*332.0*	N/A	352.0	1259.0	2433.0
Sequence 6.3	285.0	N/A	**194.0**	N/A	365.0	757.0	1676.0
Sequence 7.1	522.0	N/A	**282.0**	N/A	355.0	727.0	956.0
Sequence 7.2	**124.0**	N/A	135.0	N/A	149.0	383.0	458.0
Sequence 7.3	317.0	N/A	493.0	N/A	*293.0*	483.0	1124.0
Sequence 8.1	307.0	N/A	**291.0**	N/A	422.0	619.0	1019.0
Sequence 8.2	193.0	N/A	**180.0**	N/A	271.0	857.0	838.0
Sequence 8.3	*279.0*	N/A	353.0	N/A	305.0	1081.0	946.0

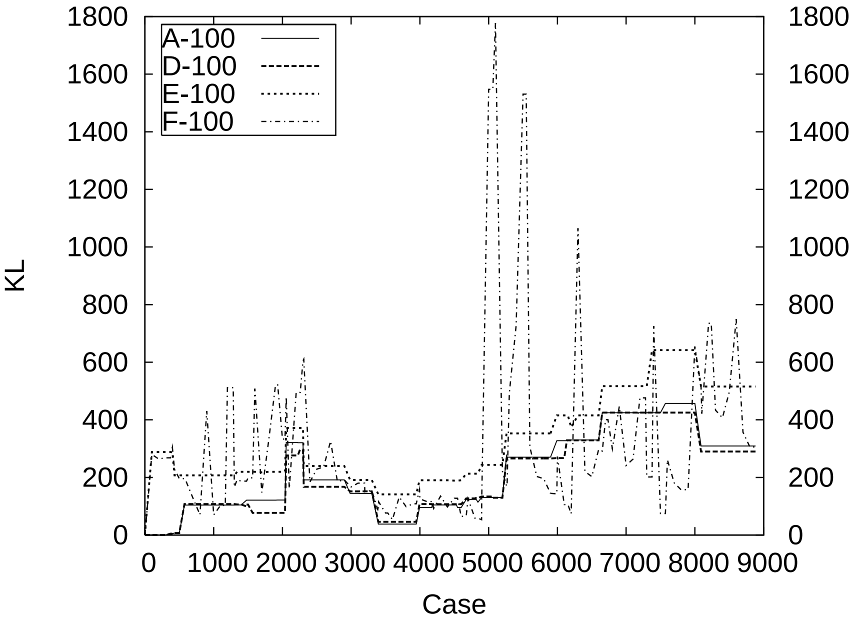


Figure 13: *KL-distance between generating distribution and learned distribution for each case in an experiment based on Sequence 1.1.*

Table 3: *dev(KL) when starting with correct network.*

	A-300	B-300	C-300	D-300	E-300	F-300	G-300
Sequence 1.1	88.7	82.7	104.0	**67.6**	241.0	158.0	215.0
Sequence 1.2	53.0	43.0	54.1	**39.9**	167.0	89.8	82.0
Sequence 1.3	71.5	93.4	78.7	**70.2**	197.0	111.0	118.0
Sequence 1.4	98.8	105.0	99.8	**90.4**	237.0	136.0	150.0
Sequence 1.5	132.0	106.0	111.0	**92.5**	260.0	124.0	129.0
Sequence 2.1	40.2	31.5	39.8	44.4	130.0	*29.8*	50.1
Sequence 2.2	71.3	73.0	82.2	**63.5**	294.0	131.0	170.0
Sequence 2.3	99.8	**85.2**	122.0	102.0	315.0	153.0	185.0
Sequence 2.4	55.4	*45.8*	56.3	45.9	194.0	55.3	60.5
Sequence 2.5	52.6	**25.6**	40.3	36.9	160.0	43.4	63.7
Sequence 3.1	170.0	139.0	164.0	150.0	346.0	**101.0**	232.0
Sequence 3.2	190.0	218.0	196.0	172.0	572.0	*159.0*	269.0
Sequence 3.3	33.2	29.9	33.1	31.0	140.0	*21.2*	31.1
Sequence 3.4	34.4	31.5	33.8	28.3	80.1	**16.7**	26.0
Sequence 3.5	175.0	142.0	178.0	*102.0*	397.0	119.0	180.0
Sequence 4.1	117.0	*86.8*	95.8	96.3	241.0	115.0	121.0
Sequence 4.2	112.0	103.0	154.0	*98.7*	289.0	115.0	158.0
Sequence 4.3	59.3	**43.3**	53.4	54.2	187.0	82.6	85.5
Sequence 4.4	77.9	68.9	82.5	*62.2*	273.0	74.0	135.0
Sequence 4.5	59.1	*42.3*	70.8	52.9	190.0	64.6	97.0
Sequence 5.1	201.0	N/A	*196.0*	N/A	205.0	1653.0	840.0
Sequence 5.2	165.0	N/A	**163.0**	N/A	267.0	600.0	652.0
Sequence 5.3	235.0	N/A	**183.0**	N/A	299.0	468.0	843.0
Sequence 6.1	1469.0	N/A	1678.0	N/A	**418.0**	1299.0	1117.0
Sequence 6.2	329.0	N/A	*286.0*	N/A	338.0	811.0	2370.0
Sequence 6.3	**155.0**	N/A	213.0	N/A	350.0	1322.0	1631.0
Sequence 7.1	277.0	N/A	**255.0**	N/A	339.0	486.0	837.0
Sequence 7.2	**90.5**	N/A	294.0	N/A	140.0	447.0	455.0
Sequence 7.3	240.0	N/A	654.0	N/A	278.0	*216.0*	857.0
Sequence 8.1	*324.0*	N/A	2607.0	N/A	401.0	1382.0	1331.0
Sequence 8.2	*166.0*	N/A	227.0	N/A	261.0	228.0	995.0
Sequence 8.3	*259.0*	N/A	446.0	N/A	293.0	1901.0	2258.0

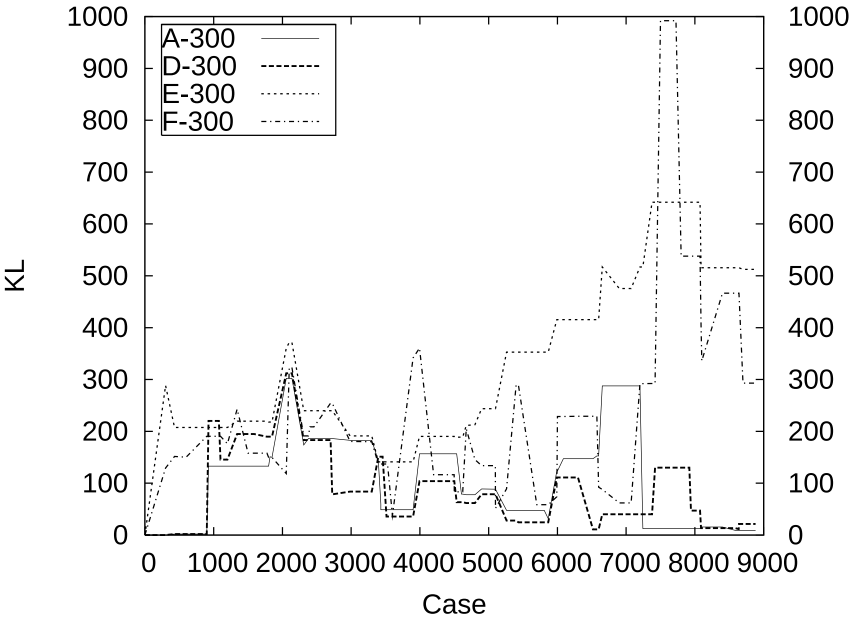


Figure 14: *KL-distance between generating distribution and learned distribution for each case in an experiment based on Sequence 1.1.*

Table 4: *dev(KL) when starting with alternative network.*

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

the methods start with the correct network and then proceeds to change it over time. This can be verified by studying Table 6, 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 boundary 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.

Comparing efficiencies, 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. It seems to be a general trend that Methods A to D perform best for k equal to 100, but loses the edge when we increase k. All methods do tend to increase in efficiency when k is increased, though. The boost for Methods A to D is simply less than for F and G. This can be because these latter methods are now forced to learn more seldom, and the generating distributions can thus change more drastically from one learning run to the next, and hence the potential for decrease in KL-distance increases.

Moving to considering structural differences, the overall picture is rather different than the one for KL-distances: Method E is now clearly the better method wrt deviance of WV and WC, as can be seen from Tables 7 and 8. This is perhaps not so surprising as the method is the method with the loosest binding to the current network, and as such can be seen to learn a new graph every kth case. This makes the method less vulnerable to poorly learned nets earlier in the experiment, unlike Methods F and G (which have a kind of momentum built into the set of sufficient statistics) and Methods A to D (which can only recover from a poorly learned graph fragment by getting a drop in conflict measures at that node later on). That Method E is not inherently superior at learning can be confirmed from noting its efficiency scores wrt WC, where it is the worst.

In general, Methods A to D are performing relatively poorly wrt deviance of WV and WC. By studying the tables it appears that deviance of Methods A to D even soars when moving from k being 100 to 300. This can be explained by the fact that the quarantine mechanism in Methods A to D causes these to abstain from relearning a graph fragment for a node X for the

Table 5: *eff(KL) when starting with correct network.*

	A-300	B-300	C-300	D-300	E-300	F-300	G-300
Sequence 1.1	-6.33	2.29	0.38	*5.63*	-27.2	-10.7	-7.17
Sequence 1.2	-2.93	-1.68	-2.56	*1.05*	-19.9	-4.49	-4.04
Sequence 1.3	-8.80	-3.28	-7.27	-1.18	-12.7	*3.19*	-10.8
Sequence 1.4	8.16	7.85	9.29	3.81	-13.9	*14.3*	-13.5
Sequence 1.5	**3.10**	-4.23	-1.03	-4.18	-32.1	-8.94	-14.5
Sequence 2.1	*5.07*	3.18	-0.00	4.33	-3.07	0.32	1.68
Sequence 2.2	-3.59	-6.85	8.06	3.50	-28.0	7.87	*13.8*
Sequence 2.3	6.35	0.60	*6.97*	-14.2	-18.9	-13.1	1.70
Sequence 2.4	-6.57	-7.24	-7.74	-5.41	-13.7	*-3.08*	-4.35
Sequence 2.5	0.74	**6.27**	-0.96	0.37	-2.36	0.31	-4.10
Sequence 3.1	17.5	19.1	26.7	**38.7**	-24.8	-7.76	-28.2
Sequence 3.2	-35.7	-31.1	-14.3	-20.0	-45.4	-1.73	*11.6*
Sequence 3.3	4.85	**6.01**	3.90	4.29	-9.88	0.65	-1.08
Sequence 3.4	0.80	*3.04*	0.86	1.44	-7.11	1.63	1.33
Sequence 3.5	23.8	1.80	*27.3*	6.69	-12.5	4.52	-5.84
Sequence 4.1	-16.6	-22.8	-8.03	-7.50	-20.4	**4.03**	-5.25
Sequence 4.2	*3.63*	-19.2	-9.89	1.59	-29.4	-5.47	-13.1
Sequence 4.3	-2.73	*6.07*	4.66	0.67	-8.36	-0.77	0.59
Sequence 4.4	2.22	12.1	-2.53	**19.2**	-45.2	-4.05	-7.58
Sequence 4.5	-6.59	**16.9**	-0.63	8.89	-27.2	-5.47	-6.77
Sequence 5.1	82.0	N/A	39.9	N/A	-14.8	*1234.0*	9.29
Sequence 5.2	-41.8	N/A	-69.4	N/A	-14.7	-75.8	*24.0*
Sequence 5.3	15.7	N/A	-44.0	N/A	-18.1	-26.2	*130.0*
Sequence 6.1	247.0	N/A	**404.0**	N/A	-37.7	-148.0	-45.3
Sequence 6.2	99.1	N/A	-9.89	N/A	-17.5	174.0	**1157.0**
Sequence 6.3	-0.43	N/A	-51.7	N/A	-20.5	-3.97	**403.0**
Sequence 7.1	10.2	N/A	1.88	N/A	-21.1	52.4	*281.0*
Sequence 7.2	19.3	N/A	-73.5	N/A	-4.85	**150.0**	60.4
Sequence 7.3	39.2	N/A	*402.0*	N/A	-12.1	75.4	5.49
Sequence 8.1	13.3	N/A	**1521.0**	N/A	-41.0	119.0	-90.5
Sequence 8.2	1.09	N/A	34.0	N/A	-27.5	15.1	*211.0*
Sequence 8.3	67.4	N/A	7.67	N/A	-26.0	484.0	*585.0*

Table 6: *eff(KL) when starting with alternative network.*

	A-300	B-300	C-300	D-300	E-300	F-300	G-300
Sequence 1.1	26.2	17.8	25.7	**29.6**	-27.2	-8.09	-10.3
Sequence 1.2	10.2	11.8	**17.5**	12.4	-22.0	-2.27	-5.66
Sequence 1.3	-4.63	-5.96	5.89	**9.06**	-12.7	-3.09	-3.67
Sequence 1.4	19.5	18.6	**22.5**	20.1	-12.0	-3.74	-11.6
Sequence 1.5	*0.23*	-5.31	-15.5	-3.46	-32.1	-6.72	-13.1
Sequence 2.1	*16.2*	10.8	7.82	8.31	-3.07	-0.92	2.60
Sequence 2.2	9.88	8.62	15.9	9.77	-28.0	6.30	*16.4*
Sequence 2.3	-14.1	-4.13	-2.85	*15.4*	-17.7	-5.70	-9.36
Sequence 2.4	**5.30**	-4.04	0.25	1.07	-12.7	-3.21	-2.21
Sequence 2.5	0.55	1.38	2.51	*3.23*	-2.22	0.39	-2.40
Sequence 3.1	46.5	**56.3**	33.7	40.5	-21.1	2.49	-19.3
Sequence 3.2	-11.4	-17.2	**49.0**	15.9	-45.4	-20.0	-33.2
Sequence 3.3	9.57	7.73	9.28	**10.9**	-9.88	0.58	-0.15
Sequence 3.4	8.47	8.17	**12.6**	7.69	-6.15	1.90	0.94
Sequence 3.5	*28.3*	4.23	15.0	20.0	-12.5	1.97	-9.97
Sequence 4.1	-9.05	-0.10	-8.77	2.79	-20.4	2.33	*11.8*
Sequence 4.2	-14.7	2.28	-5.30	**8.98**	-26.2	-7.26	-10.1
Sequence 4.3	-3.73	-2.00	-2.35	*3.51*	-7.29	-4.50	3.24
Sequence 4.4	-9.05	*6.50*	-12.6	3.88	-45.2	-6.61	-8.41
Sequence 4.5	-12.8	-4.60	-1.53	*11.5*	-25.8	-0.05	-5.78

Table 7: *dev(WV) when starting with correct network.*

	A-300	B-300	C-300	D-300	E-300	F-300	G-300
Sequence 1.1	9.47	11.3	9.92	11.6	**7.15**	7.33	7.44
Sequence 1.2	10.0	13.2	9.86	12.4	**6.82**	7.32	7.44
Sequence 1.3	10.9	13.1	10.9	12.5	**5.30**	5.85	5.84
Sequence 1.4	9.06	10.9	9.10	11.1	**5.63**	5.86	6.01
Sequence 1.5	8.13	10.4	8.93	10.6	**4.00**	4.54	4.62
Sequence 2.1	11.4	12.1	10.6	12.3	**6.42**	7.37	7.99
Sequence 2.2	11.7	13.7	11.3	12.6	**7.53**	7.88	8.18
Sequence 2.3	12.1	13.6	12.4	12.6	**6.60**	6.88	7.02
Sequence 2.4	15.2	17.0	15.0	16.5	**6.21**	7.11	7.35
Sequence 2.5	13.4	13.8	12.4	15.4	**5.50**	6.44	6.54
Sequence 3.1	11.4	11.6	11.5	11.3	**4.57**	5.42	5.71
Sequence 3.2	9.61	10.6	9.61	9.46	4.53	**4.21**	5.49
Sequence 3.3	13.0	13.1	12.8	13.8	**6.32**	7.92	7.98
Sequence 3.4	15.3	15.3	14.8	15.5	**5.92**	7.93	8.53
Sequence 3.5	9.83	11.0	9.95	10.3	**4.71**	5.55	5.64
Sequence 4.1	11.8	12.6	10.8	12.7	**6.53**	7.07	7.36
Sequence 4.2	10.6	11.6	10.7	13.2	**7.27**	7.33	7.68
Sequence 4.3	10.8	12.7	10.9	12.2	**7.29**	8.37	8.18
Sequence 4.4	8.41	9.57	7.76	9.91	**5.15**	6.04	5.70
Sequence 4.5	8.03	10.3	8.33	10.8	**5.70**	6.09	6.41
Sequence 5.1	22.1	N/A	20.4	N/A	**11.1**	13.0	13.0
Sequence 5.2	23.6	N/A	23.4	N/A	**11.1**	11.6	12.2
Sequence 5.3	25.1	N/A	23.7	N/A	**11.1**	12.5	12.3
Sequence 6.1	27.9	N/A	28.2	N/A	**12.0**	13.6	13.6
Sequence 6.2	25.3	N/A	25.2	N/A	**11.9**	13.3	13.8
Sequence 6.3	30.8	N/A	32.9	N/A	**11.3**	12.8	13.2
Sequence 7.1	38.8	N/A	37.4	N/A	**10.7**	12.5	13.6
Sequence 7.2	47.2	N/A	47.9	N/A	**10.7**	15.4	16.6
Sequence 7.3	45.9	N/A	43.8	N/A	**11.5**	14.1	15.5
Sequence 8.1	18.5	N/A	21.3	N/A	**7.91**	9.51	9.06
Sequence 8.2	27.1	N/A	27.1	N/A	**10.6**	12.9	12.4
Sequence 8.3	24.5	N/A	25.2	N/A	**12.0**	13.1	13.8

next $2k$ cases, every time they have learned a new graph fragment for X. This means that, if there are less than $2k$ cases between two shifts involving the same node, then the change is ignored. When k grows to 300, we have a minimum required gap of 600 observations between two shifts involving the same node. As the number of cases between any two shifts is between 200 and 1000 for Settings 1 to 4, the chance that this never happens is small. The effect gets even more protrusive for experiments involving Settings 5 to 8, where the number of cases between any two shifts can go as low as 50.

Methods A to D does perform convincingly wrt efficiency of WC, but less so wrt efficiency of WV (as can be seen from Tables 9 and 10). This difference indicates that either our heuristics are too poor at detecting changes in v-structures, or that we fail to identify some separating sets in LearnFragments(·). More elaborate experiments are needed to clarify this. Interestingly, the efficiency wrt WV gets competitive with the other methods when the start network is not the correct one. Currently, we have no explanation for this phenomena.

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. Here the performance is not satisfactory. The reasonable efficiency statistics indicate that this must be because we fail to identify all changed nodes in a shift, which brings us to the second question that unfortunately must be answered in the positive: Either the heuristics are too poor, or the algorithm must be changed to take more nodes into account. Most pressingly, it seems like destruction and creation

Table 8: *dev(WC) when starting with correct network.*

	A-300	B-300	C-300	D-300	E-300	F-300	G-300
Sequence 1.1	9.47	11.3	9.92	11.6	**7.15**	7.33	7.44
Sequence 1.2	10.0	13.2	9.86	12.4	**6.82**	7.32	7.44
Sequence 1.3	10.9	13.1	10.9	12.5	**5.30**	5.85	5.84
Sequence 1.4	9.06	10.9	9.10	11.1	**5.63**	5.86	6.01
Sequence 1.5	8.13	10.4	8.93	10.6	**4.00**	4.54	4.62
Sequence 2.1	11.4	12.1	10.6	12.3	**6.42**	7.37	7.99
Sequence 2.2	11.7	13.7	11.3	12.6	**7.53**	7.88	8.18
Sequence 2.3	12.1	13.6	12.4	12.6	**6.60**	6.88	7.02
Sequence 2.4	15.2	17.0	15.0	16.5	**6.21**	7.11	7.35
Sequence 2.5	13.4	13.8	12.4	15.4	**5.50**	6.44	6.54
Sequence 3.1	11.4	11.6	11.5	11.3	**4.57**	5.42	5.71
Sequence 3.2	9.61	10.6	9.61	9.46	4.53	**4.21**	5.49
Sequence 3.3	13.0	13.1	12.8	13.8	**6.32**	7.92	7.98
Sequence 3.4	15.3	15.3	14.8	15.5	**5.92**	7.93	8.53
Sequence 3.5	9.83	11.0	9.95	10.3	**4.71**	5.55	5.64
Sequence 4.1	11.8	12.6	10.8	12.7	**6.53**	7.07	7.36
Sequence 4.2	10.6	11.6	10.7	13.2	**7.27**	7.33	7.68
Sequence 4.3	10.8	12.7	10.9	12.2	**7.29**	8.37	8.18
Sequence 4.4	8.41	9.57	7.76	9.91	**5.15**	6.04	5.70
Sequence 4.5	8.03	10.3	8.33	10.8	**5.70**	6.09	6.41
Sequence 5.1	22.1	N/A	20.4	N/A	**11.1**	13.0	13.0
Sequence 5.2	23.6	N/A	23.4	N/A	**11.1**	11.6	12.2
Sequence 5.3	25.1	N/A	23.7	N/A	**11.1**	12.5	12.3
Sequence 6.1	27.9	N/A	28.2	N/A	**12.0**	13.6	13.6
Sequence 6.2	25.3	N/A	25.2	N/A	**11.9**	13.3	13.8
Sequence 6.3	30.8	N/A	32.9	N/A	**11.3**	12.8	13.2
Sequence 7.1	38.8	N/A	37.4	N/A	**10.7**	12.5	13.6
Sequence 7.2	47.2	N/A	47.9	N/A	**10.7**	15.4	16.6
Sequence 7.3	45.9	N/A	43.8	N/A	**11.5**	14.1	15.5
Sequence 8.1	18.5	N/A	21.3	N/A	**7.91**	9.51	9.06
Sequence 8.2	27.1	N/A	27.1	N/A	**10.6**	12.9	12.4
Sequence 8.3	24.5	N/A	25.2	N/A	**12.0**	13.1	13.8

Table 9: *eff(WV) when starting with correct network.*

	A-300	B-300	C-300	D-300	E-300	F-300	G-300
Sequence 1.1	-0.36	-0.40	-0.44	-0.66	-0.8	**-0.15**	-0.19
Sequence 1.2	-0.53	-0.34	-0.70	-0.50	-0.84	*-0.16*	-0.26
Sequence 1.3	-0.95	-1.26	-0.81	-1.1	-0.51	**-0.24**	-0.29
Sequence 1.4	-0.67	-0.62	-0.39	-0.41	-0.32	**-0.02**	-0.04
Sequence 1.5	-0.45	-0.90	-0.29	-0.39	-0.38	-0.15	**-0.12**
Sequence 2.1	-0.43	-0.86	-0.34	-0.76	-0.34	-0.08	**-0.02**
Sequence 2.2	-0.27	-0.71	-0.40	-0.39	-0.54	*-0.09*	-0.22
Sequence 2.3	-0.29	-0.67	-0.33	-0.56	-0.24	*-0.09*	-0.10
Sequence 2.4	-1.03	-1.98	-1.54	-2.02	-0.50	-0.25	**-0.24**
Sequence 2.5	-0.62	-1.27	-0.81	-1.27	-0.21	-0.04	**-0.01**
Sequence 3.1	-1.46	-1.01	-1.25	-1.44	-0.44	**0.13**	-0.13
Sequence 3.2	-0.38	-0.35	-0.47	-0.41	-0.32	**0.08**	-0.13
Sequence 3.3	-0.90	-0.88	-1.02	-1.08	-0.39	**0.14**	-0.07
Sequence 3.4	-1.19	-1.13	-1.61	-1.38	-0.25	**0.23**	0.22
Sequence 3.5	-0.74	-0.59	-0.77	-0.28	-0.25	*-0.18*	-0.25
Sequence 4.1	-0.55	-0.71	-0.14	-0.48	-0.64	*-0.08*	-0.19
Sequence 4.2	-0.43	-0.60	-0.39	-0.40	-0.64	**-0.14**	-0.16
Sequence 4.3	-0.46	-0.55	-0.50	-0.58	-0.33	**0.01**	-0.04
Sequence 4.4	-0.36	-0.47	-0.35	-0.61	-0.33	-0.10	**-0.06**
Sequence 4.5	-0.19	-0.13	*0.05*	-0.10	-0.49	-0.02	-0.18
Sequence 5.1	-1.08	N/A	-0.76	N/A	-0.58	**-0.01**	-0.36
Sequence 5.2	-1.95	N/A	-1.23	N/A	-0.76	**-0.14**	-0.49
Sequence 5.3	-2.17	N/A	-1.64	N/A	-0.50	**-0.14**	-0.29
Sequence 6.1	-2.56	N/A	-2.21	N/A	-1.35	**-0.13**	-0.48
Sequence 6.2	-2.13	N/A	-2.2	N/A	-0.72	**-0.10**	-0.51
Sequence 6.3	-0.22	N/A	**0.30**	N/A	-0.57	-0.22	-0.36
Sequence 7.1	-2.96	N/A	-2.84	N/A	-0.48	**0.18**	-0.09
Sequence 7.2	-4.20	N/A	-6.21	N/A	-0.45	**0.40**	-0.38
Sequence 7.3	-4.67	N/A	-4.64	N/A	-0.63	**0.06**	-0.20
Sequence 8.1	-1.86	N/A	-2.03	N/A	-0.47	**-0.05**	-0.18
Sequence 8.2	-2.53	N/A	-2.73	N/A	-0.88	**-0.05**	-0.40
Sequence 8.3	-1.95	N/A	-2.0	N/A	-0.57	**0.14**	-0.26

Table 10: *eff(WC) when starting with correct network.*

	A-300	B-300	C-300	D-300	E-300	F-300	G-300
Sequence 1.1	2.18	**4.43**	2.91	4.20	-7.88	-2.31	-3.50
Sequence 1.2	3.63	**6.79**	3.05	5.99	-7.68	-1.17	-2.20
Sequence 1.3	2.94	**4.46**	1.97	4.18	-10.9	-3.69	-4.53
Sequence 1.4	0.71	2.39	1.31	**3.05**	-9.36	-4.90	-5.30
Sequence 1.5	-0.22	2.06	0.42	**3.52**	-11.4	-5.76	-5.88
Sequence 2.1	3.18	3.76	2.87	**3.82**	-8.27	-0.68	-1.76
Sequence 2.2	5.65	**6.44**	4.19	5.05	-6.73	-0.66	-1.60
Sequence 2.3	4.84	5.67	**5.81**	4.46	-7.61	-2.64	-3.33
Sequence 2.4	7.59	**8.17**	6.33	7.61	-9.80	-1.78	-2.70
Sequence 2.5	3.25	2.79	1.66	**4.05**	-8.75	-1.72	-2.66
Sequence 3.1	3.42	2.99	**3.48**	2.22	-10.6	-2.25	-3.32
Sequence 3.2	1.75	**3.48**	1.68	0.76	-10.4	-1.25	-3.21
Sequence 3.3	3.56	3.80	3.89	**4.32**	-8.83	-0.44	-1.98
Sequence 3.4	**5.28**	4.32	4.96	4.54	-8.89	1.31	-0.52
Sequence 3.5	1.31	**3.10**	2.41	2.44	-8.96	-1.01	-2.57
Sequence 4.1	2.33	2.48	2.52	**3.00**	-9.43	-1.86	-2.88
Sequence 4.2	3.27	3.71	3.42	**4.22**	-8.45	-1.66	-2.94
Sequence 4.3	2.96	**4.89**	3.01	3.55	-7.13	-0.87	-1.38
Sequence 4.4	1.03	**1.65**	0.66	1.49	-10.9	-5.32	-5.89
Sequence 4.5	1.40	3.25	2.00	**3.31**	-7.58	-1.57	-2.05
Sequence 5.1	7.27	N/A	**7.79**	N/A	-13.7	-5.09	-9.04
Sequence 5.2	**9.15**	N/A	9.04	N/A	-16.1	-7.70	-11.9
Sequence 5.3	**11.2**	N/A	10.0	N/A	-16.1	-7.53	-12.2
Sequence 6.1	**16.3**	N/A	15.8	N/A	-17.2	-6.65	-11.0
Sequence 6.2	8.35	N/A	**9.70**	N/A	-14.6	-6.14	-10.3
Sequence 6.3	11.5	N/A	**13.2**	N/A	-15.3	-7.38	-11.0
Sequence 7.1	**21.8**	N/A	20.0	N/A	-17.3	-4.68	-9.09
Sequence 7.2	21.0	N/A	**23.9**	N/A	-14.3	-1.82	-8.48
Sequence 7.3	**24.7**	N/A	21.4	N/A	-15.7	-3.73	-7.54
Sequence 8.1	1.19	N/A	**4.12**	N/A	-19.9	-12.1	-15.0
Sequence 8.2	**12.9**	N/A	12.1	N/A	-17.5	-5.80	-11.2
Sequence 8.3	**12.6**	N/A	12.2	N/A	-14.8	-7.10	-9.94

of v-structures in the underlying generating distribution through arc reversals (as opposed to arc insertion or removal) fails to be detected. More focused experiments are needed to clarify this issue. Finally, the third 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.

Conclusion

The method that we have presented solves the problem of keeping a BN model updated in face of new observations. The distinguishing features are the two main ideas of only learning when new observations are highly unlikely given the current model, and in those cases only tampering with the parts of the model that are contested by evidence.

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 since our experiments indicate that the ones we have chosen do not perform extremely well, new heuristics might be evaluated in isolation by testing empirically how well they live up to these expectations.

Currently, we have a series of ideas for optimising 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, as the measures score-based approaches optimise 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.

Caprile, B. (1999). Uniformly generating distribution functions for discrete random variables. Technical report, ITC-irst.

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.

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.

Koster, A. M. C. A., H. L. Bodlaender, and S. P. M. van Hoesel (2001). Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics 8*, 1–24.

Lam, W. (1998). Bayesian network refinement via machine learning approach. *IEEE Transactions on Pattern Analasys 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.

Lauritzen, S. L., A. P. Dawid, B. N. Larsen, and H.-G. Leimer (1990). Independence properties in Markov fields. *Networks 20*(5), 491–505.

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, 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 (2001). *Causation, prediction, and search* (2nd ed.). MIT Press.

Szwarcfiter, J. L. and P. E. Lauer (1976). A search strategy for the elementary cycles of a directed graph. *BIT Numerical Mathematics 16*(2), 192–204.

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