Learning Bayesian networks with low inference complexity

Marco Benjumeda[1] · **Pedro Larrañaga**[1] · **Concha Bielza**[1]

Abstract One of the main research topics in machine learning nowadays is the improvement of the inference and learning processes in probabilistic graphical models. Traditionally, inference and learning have been treated separately, but given that the structure of the model conditions the inference complexity, most learning methods will sometimes produce inefficient inference models. In this paper we propose a framework for learning low inference complexity Bayesian networks. For that, we use a representation of the network factorization that allows efficiently evaluating an upper bound in the inference complexity of each model during the learning process. Experimental results show that the proposed methods obtain tractable models that improve the accuracy of the predictions provided by approximate inference in models obtained with a well-known Bayesian network learner.

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness through the Cajal Blue Brain (C080020-09; the Spanish partner of the Blue Brain initiative from EPFL) and TIN2013-41592-P projects, by the Regional Government of Madrid through the S2013/ICE-2845-CASI-CAM-CM project, and by the European Union's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement No. 604102 (Human Brain Project). M. Benjumeda is supported by a predoctoral contract for the formation of doctors from the Spanish Ministry of Economy and Competitiveness (BES-2014-068637).

✉ Marco Benjumeda
marcobb8@gmail.com

Pedro Larrañaga
pedro.larranaga@fi.upm.es

Concha Bielza
mcbielza@fi.upm.es

[1] Computational Intelligence Group, Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid, Madrid, Spain

1 Introduction

Bayesian networks (BNs) are very powerful tools for concisely modeling probability distributions of a set of random variables $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$, and have also been used for supervised classification [28] and clustering [7]. A BN $\mathcal{B}$ encodes the conditional dependences between the variables in $\mathcal{X}$, and it is composed of:

1. *Directed acyclic graph* Each node of the graph represents a random variable in $\mathcal{X}$, and the arcs represent the probabilistic dependences among variables.
2. *Parameters* There is a conditional probability distribution (CPD) $P(X_i|\mathbf{pa}_{\mathcal{B}}(X_i))$ associated to each variable $X_i \in \mathcal{X}$, where $\mathbf{pa}_{\mathcal{B}}(X_i)$ are the parents of X_i in $\mathcal{B}$. These CPDs are called the parameters of the network.

In the past years there has been a huge interest in the creation of new methods for learning the structure of BNs from data. Usually, two types of scores are used. Bayesian metrics maximize the posterior probability of the network given a prior distribution over all the possible networks conditioned to the data, while information theory metrics try to maximize the data compression achieved by each network. Well known scoring functions such as Bayesian Dirichlet equivalent (BDe) [20], K2 [13,14,25,26], Akaike information criterion (AIC) [1], Bayesian information criterion (BIC) [31] or minimum description length (MDL) [10,24] implicitly or explicitly penalize the representation complexity of the network using the number of parameters of the model. Nev-

ertheless, the representation complexity and the inference complexity are sometimes very different for the same model [5], so an indicator of the inference complexity is required to learn models where inference is tractable.

In this work we propose a new framework for learning low inference complexity BNs. For that, we incrementally compile a complementary structure that we use to compute an upper bound in the inference complexity of each network efficiently.

This paper is organized as follows. The rest of Sect. 1 provides an introduction to the problem of learning low inference complexity models. Section 2 gives a brief review of network polynomials (NPLs), the base of our work. Section 3 describes the representation and the methods proposed in this paper for learning learning tractable BNs. Section 4 shows the experimental results. Section 5 gives the conclusive remarks and future research lines.

1.1 Learning thin models

Exact inference in BNs is NP-hard [12]. We can perform evidence propagation in linear time in the number of variables in a BN when the topology of the network is a polytree [22]. Nevertheless, this constrain is too hard in many situations where polytrees do not have enough representative power.

Approximate inference is commonly used when exact inference is intractable. Approximate inference in BNs is also NP-hard [15], and although it has been useful for solving some otherwise intractable problems it has some relevant flaws. It produces a worsening in the answers provided by the model, and it is challenging to check the convergence of the algorithms.

The complexity of a BN depends mostly on its structure. Exact inference is exponential in the treewidth of the network, which is the size of the biggest node in the minimal tree decomposition of the network minus one [9]. Intuitively, it represents how similar is the network structure to a tree.

Computing the treewidth of a graph exactly is an NP-complete problem [3], and approximate methods for treewidth estimation are also computationally expensive. On the other hand, it is possible to check if the treewidth of a graph is less or equal than a fixed k in linear time in the number of variables [9].

There are some approaches that use a bound on the treewidth to learn thin models [4,11,17,30]. Nevertheless, checking if the treewidth is less or equal than k is super-exponential in k [8], so in practice it is intractable when k is not very small.

Lowd and Domingos [27] used the incremental compilation of arithmetic circuits (see Sect. 2.1) to learn tractable models that exploit the local structures of the networks, penalizing the size of the circuits instead of bounding the treewidth.

2 Network polynomials

The probability distribution implicit in any BN $\mathcal{B}$ can be also represented as a network polynomial [16], that is, a multilinear function over two types of variables, indicators I_{x_i} and parameters $\theta_{x_i|\pi_i} = P(X_i = x_i|\mathbf{pa}_{\mathcal{B}}(X_i) = \pi_i)$. The indicators are Boolean functions that return 1 if $X_i = x_i$ or if the value of X_i is unknown, and 0 otherwise. The probability distribution of discrete variables $X_1, \ldots, X_n$ in an NPL is described as:

$$P(X_1 = x_1, \ldots, X_n = x_n) = \sum_{i=1}^{n} \prod_{\substack{x_i \in \Omega_{X_i} \\ \pi_i \in \Omega_{\mathbf{pa}_{\mathcal{B}}(X_i)}}} I_{x_i} \theta_{x_i|\pi_i}$$

where we use $x_i \in \Omega_{X_i}$ to represent each configuration x_i of variable X_i, and $\pi_i \in \Omega_{\mathbf{pa}_{\mathcal{B}}(X_i)}$ to represent each configuration π_i of the parents of X_i.

This function represents the joint probability of the set of variables $X_1, \ldots, X_n$. NPLs allow answering any arbitrary marginal or conditional probabilistic query in linear time in the size (number of sums and products) of the polynomial.

2.1 Arithmetic circuits

The size of an NPL grows exponentially with the number of variables, making inference nearly intractable for common-size networks. Trying to overcome this difficulty, Darwiche [16] proposed the use of ACs to represent NPLs, using the distributive properties of the polynomials to reduce the complexity of the NPL function.

ACs are directed acyclic graphs (DAGs) in which the inner nodes (nodes with children) are addition ($+$) and multiplication ($\times$) nodes and the leaves (nodes without children) are numeric variables or constants. Performing inference in the circuits is straightforward and linear in the number of arcs of the graph. The circuit can be evaluated bottom-up by computing the operations represented by each inner node ($+$, $\times$) from the values of its children, starting from the leaves.

For example, the AC shown in Fig. 1 represents the NPL shown in Eq. (1). In the examples shown in this paper we only use Boolean variables, and for each variable X_i we will refer to $X_i = \text{True}$ as x_i and $X_i = \text{False}$ as $\neg x_i$.

$$\begin{aligned} P(A, B) = {} & I_a I_b \theta_a \theta_{b|a} + I_a I_{\neg b} \theta_a \theta_{\neg b|a} + I_{\neg a} I_b \theta_{\neg a} \theta_{b|\neg a} \\ & + I_{\neg a} I_{\neg b} \theta_{\neg a} \theta_{\neg b|\neg a} \ . \end{aligned} \tag{1}$$

Darwiche [16] uses ACs as a complementary representation obtained by the compilation of BNs to perform tractable exact inference. Lowd and Domingos proposed the method LearnAC [27], that is the first approach to learn ACs directly from data. It uses a greedy search process that penalizes each

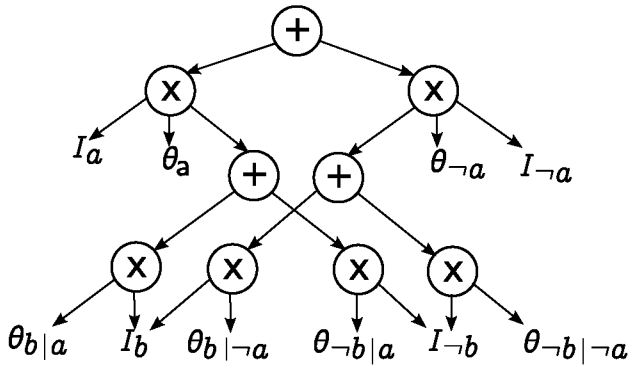


Fig. 1 AC representation for the NPL of Eq. (1)

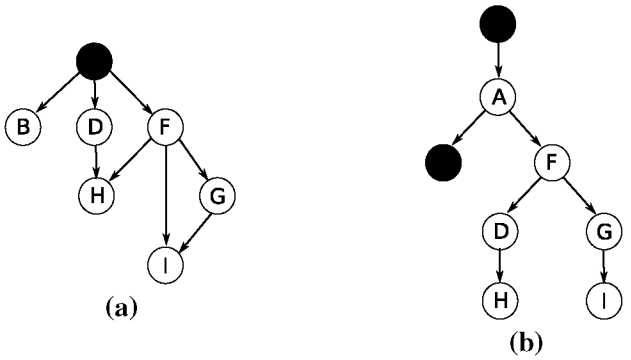


Fig. 2 **a** Structure of a BN, and **b** a PT associated to the BN

circuit with its size, given by the number of arcs of the circuit. The changes produced by LearnAC during the learning process consist of splitting the leafs of the circuit, which allows exploiting the local structures of the network.

The size of the circuits and the number of possible local changes that LearnAC must consider in each iteration can be huge when the number of variables of the models is not very small, making the search expensive.

3 Polynomial trees

Some local changes that are usually considered in most score+search BN learning methods, such as arc removals or reversions, are not considered by LearnAC or any method for learning bounded treewidth junction trees (JTs), and they are essential for avoiding local optima in early stages of the learning process. We use a simpler representation, called polynomial trees (PTs), to consider these operations, allowing a more flexible learning process of low inference complexity models.

A PT is a compact graphical representation of an NPL. For simplicity, we also maintain the BN representation to show the conditional dependences between the variables of the model. Each PT is associated to a BN, and it represents a factorization of the NPL encoded by the BN. A PT $\mathcal{P}$ associated to a BN $\mathcal{B}$ over a set of variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ consists of:

1. A set of nodes $\mathcal{X}_P = \{*\} \cup \mathcal{X}$, where $*$ is the root node.
2. An indicator set $\mathcal{I} = \{I(X_i), \ldots, I(X_n)\}$, where each indicator $I(X_i) \in \mathcal{I}$ can take any value of X_i or the value $\varnothing$.
3. A set of directed arcs that represent a topological ordering of the nodes in $\mathcal{X}_P$, forming a tree structure.

Basically, the complete graphical representation of a PT consists of a DAG representing the dependences in the network and a tree representing a factorization of this network. Figure 2 is an example of a BN (a) and a PT associated to the BN (b).

The soundness of a PT is the property that guarantees that it can perform exact inference correctly for any probabilistic query that could be asked to the model, and it is defined as:

Definition 1 Let $\mathcal{P}$ be a PT over $\mathcal{X}_P = \{*\} \cup \mathcal{X}$ with an associated BN $\mathcal{B}$ over $\mathcal{X}$, and let $\mathbf{pred}_{\mathcal{P}}(X_i)$ be the predecessors of X_i in $\mathcal{P}$. $\mathcal{P}$ is sound for $\mathcal{B}$ if and only if $\forall X_i \in \mathcal{X}$ it holds that $\forall X_j \in \mathbf{pa}_{\mathcal{B}}(X_i)$, $X_j \in \mathbf{pred}_{\mathcal{P}}(X_i)$.

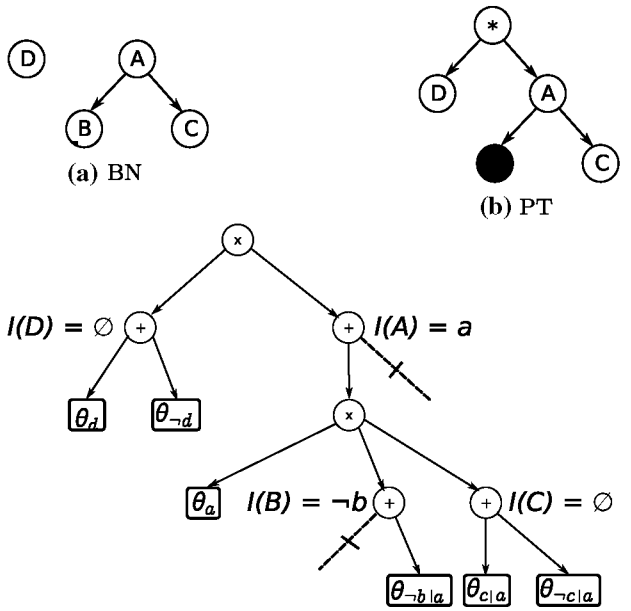
If in a PT $\mathcal{P}$ associated to a BN $\mathcal{B}$ there is at least one node X_i that has a parent X_j in $\mathcal{B}$ that does not belong to $\mathbf{pred}_{\mathcal{P}}(X_i)$, then exact inference will fail, because $I(X_j)$ will be set to $\varnothing$ when we evaluate X_i.

There are usually multiple sound PTs for each BN, because there are multiple possible orders to perform exact inference on each network. We are interested in obtaining those PTs that are sound for a BN and that are as close as possible to the PT with the smallest treewidth for this network.

3.1 Inference in polynomial trees

In this subsection we provide a method for performing exact inference in PTs. There are more efficient evidence propagation methods in the state-of-art, but our purpose is to measure the inference complexity of each PT. It proceeds by first, executing a top-down process for the propagation of the indicators in the tree, and then it performs a bottom-up process where it computes the probability of the polynomial represented in the tree given the configuration of the indicators.

Given a BN $\mathcal{B}$, a PT $\mathcal{P}$ and an evidence $\boldsymbol{e}$, the probability of $\boldsymbol{e}$ in the model can be computed as follows. First, we need to initialize the indicator set $\mathcal{I} = \{I(X_i), \ldots, I(X_n)\}$ with the values in $\boldsymbol{e}$, setting the indicators of the variables that do not appear in $\boldsymbol{e}$ to $\varnothing$. Let $\mathbf{ch}_{\mathcal{P}}(X_i)$ be the children of X_i in $\mathcal{P}$. We can evaluate the root node $*$ given $\mathcal{I}$ computing:



(c) Polynomial generated using Equation (2)

Fig. 3 Inference example. The probabilistic query $P(a, \neg b)$ is answered by the BN and its associated PT shown in **a** and **b** respectively. First, the *indicators* are set to $\mathcal{I} = \{a, \neg b, \varnothing, \varnothing\}$. The polynomial $P(a, \neg b) = (\theta_d + \theta_{\neg d}) \cdot (\theta_a \cdot \theta_{\neg b|a} \cdot (\theta_{c|a} + \theta_{\neg c|a}))$, that is shown graphically in **c** , represents the operations performed by Eq. (2) to answer this query

$$\text{query}(\mathcal{B}, \mathcal{P}, *, \mathcal{I}) = \prod_{X_k \in \mathbf{ch}_{\mathcal{P}}(*)} \text{query}(\mathcal{B}, \mathcal{P}, X_k, \mathcal{I}) \ .$$

The rest of the nodes can be recursively evaluated by computing:

$$\text{query}(\mathcal{B}, \mathcal{P}, X_i, \mathcal{I}) = \sum_{x_i \in \Omega_{C_i}} \theta_{x_i|\pi} \; X_i \cdot \prod_{X_j \in \mathbf{ch}_{\mathcal{P}}(X_i)} \text{query}(\mathcal{B}, \mathcal{P}, X_j, \mathcal{I}_{X_i}) \quad (2)$$

where $\Omega_{C_i} = \Omega_{X_i}$ if $I(X_i) = \varnothing$ and $\Omega_{C_i} = \{I(X_i)\}$ otherwise, π_{X_i} is a set with the value of each parent of X_i in $\mathcal{I}$, and $\mathcal{I}_{X_i}$ is the set of indicators obtained after setting the value of $I(X_i)$ to x_i in $\mathcal{I}$.

Figure 3 shows an example of how to perform exact inference in a simple PT to answer the probabilistic query $P(\text{a}, \neg\text{b})$.

3.2 Evaluating the complexity of polynomial trees

In most state-of-the-art methods for learning thin probabilistic models the treewidth is used as an estimation of the inference complexity. Obtaining the treewidth of a graph is an NP-complete problem, so in most methods estimations are used [5]. It is simple to obtain an upper bound in the inference complexity of a PT efficiently. The method used in this paper for the complexity evaluation of PTs obtains the maximum number of operations required to evaluate Eq. (2). It works recursively, obtaining the number of sums and products required to perform inference in each node, which is given by:

$$\text{eval}(\mathcal{P}, X_i) = |\Omega_{X_i}| \cdot \left(1 + \sum_{X_j \in \mathbf{ch}_{\mathcal{P}}(X_i)} (1 + \text{eval}(\mathcal{P}, X_j))\right) - 1. \quad (3)$$

Basically, each node X_i requires $|\Omega_{X_i}| \cdot \sum_{X_j \in \mathbf{ch}_{\mathcal{P}}(X_i)} (1 + \text{eval}(\mathcal{P}, X_j))$ operations to compute Eq. (2) for each children of X_i, multiplying the resulting values, and $|\Omega_{X_i}| - 1$ operations to sum the results for each instance of X_i.

3.3 Incremental compilation of PTs

To evaluate the inference complexity of each network using Eq. (3) it is necessary to have a compiled PT in each step of the learning process. As compiling a PT from scratch every time is intractable, we have created a group of procedures to compile incrementally any local change that could be done in a BN during the learning process, including arc additions, removals and reversals.

Let $\mathcal{P}$ be a sound PT for a BN $\mathcal{B}$, and let $\mathcal{B}'$ be the result of applying a local change to $\mathcal{B}$. It may happen that $\mathcal{P}$ is not sound for $\mathcal{B}'$. The purpose of the incremental compilation methods is to obtain a sound PT $\mathcal{P}'$ for $\mathcal{B}'$. Next, we show the procedures that we use to compile PTs incrementally.

3.3.1 Arc addition

The addition of an arc in a BN is straightforward, but in a PT the compilation process is not so simple, and it depends on the current configuration of the PT. Let us consider any addition $(X_{\text{out}} \rightarrow X_{\text{in}})$ that should be compiled in a PT $\mathcal{P}$ and included in its corresponding BN $\mathcal{B}$. We will refer to X_{out} as the output node and X_{in} as the input node. The three possible scenarios that we could face in an arc addition are the following:

1. $X_{\text{out}} \in \mathbf{pred}_{\mathcal{P}}(X_{\text{in}})$: $\mathcal{P}$ is sound for $\mathcal{B}$ after the addition, so no changes are required.
2. $X_{\text{in}} \in \mathbf{pred}_{\mathcal{P}}(X_{\text{out}})$: in this scenario, it is necessary to set X_{out} as a predecessor of X_{in} in $\mathcal{P}$ and reconfigure the positions of the nodes between X_{out} and X_{in} to obtain a PT sound for $\mathcal{B}$.
3. $X_{\text{out}} \notin \mathbf{pred}_{\mathcal{P}}(X_{\text{in}})$ and $X_{\text{in}} \notin \mathbf{pred}_{\mathcal{P}}(X_{\text{out}})$: in this case node X_{out} and its predecessors in $\mathcal{P}$ are set as predecessors of X_{in} in $\mathcal{P}$.

The procedure proposed for the incremental compilation of arc additions is described by Algorithm 1.

Data: PT $\mathcal{P}$, BN $\mathcal{B}$, output node X_{out}, input node X_{in}
Result: PT $\mathcal{P}'$, BN $\mathcal{B}'$

let $\mathcal{B}'$ and $\mathcal{P}'$ be copies of $\mathcal{B}$ and $\mathcal{P}$ respectively ;
add X_{out} to $\mathbf{pa}_{\mathcal{B}'}(X_{\text{in}})$;
if $X_{\text{in}} \in \boldsymbol{pred}_{\mathcal{P}}(X_{\text{out}})$ **then**
  let $\mathbf{desc}_{\mathcal{P}}(X_{\text{in}})$ be the descendants of X_{in} in $\mathcal{P}$;
  let $\mathcal{X}_c \leftarrow$ be the nodes in $(\mathbf{desc}_{\mathcal{P}}(X_{\text{in}}) \cap \mathbf{pred}_{\mathcal{P}}(X_{\text{out}})) \cup \{X_{\text{out}}\}$, and $\boldsymbol{X}_c$ the list obtained by ordering $\mathcal{X}_c$ from the shallowest to the deepest node ;
  $X_p \leftarrow \mathbf{pa}_{\mathcal{P}}(X_{\text{in}})$;
  $X_d \leftarrow X_{\text{in}}$;
  for $X_i \in \boldsymbol{X}_c$ **do**
    $X_m \leftarrow \mathbf{ch}_{\mathcal{P}}(X_i) \cap \mathcal{X}_c$;
    if $X_i \in \boldsymbol{desc}_{\mathcal{B}}(X_{\text{in}})$ **then**
      $\mathbf{pa}_{\mathcal{P}'}(X_i) \leftarrow X_d$;
      $X_d \leftarrow X_i$;
    else
      $\mathbf{pa}_{\mathcal{P}'}(X_i) \leftarrow X_p$;
      $X_p \leftarrow X_i$;
    end
    for $X_j \in \boldsymbol{ch}_{\mathcal{P}}(X_i) \backslash \{X_m\}$ **do**
      if $(\boldsymbol{desc}_{\mathcal{P}}(X_j) \cup \{X_j\}) \cap \boldsymbol{desc}_{\mathcal{B}}(X_{\text{in}}) \neq \varnothing$ **then**
        $\mathbf{pa}_{\mathcal{P}'}(X_j) \leftarrow X_d$;
      else
        $\mathbf{pa}_{\mathcal{P}'}(X_j) \leftarrow X_p$;
      end
    end
  end
  $\mathbf{pa}_{\mathcal{P}'}(X_{\text{in}}) \leftarrow X_{\text{out}}$;
else if $X_{\text{in}} \notin \boldsymbol{pred}_{\mathcal{P}}(X_{\text{out}})$ *and* $X_{\text{out}} \notin \boldsymbol{pred}_{\mathcal{P}}(X_{\text{in}})$ **then**
  let X_k be the shallowest node in $\mathbf{pred}_{\mathcal{P}}(X_{\text{out}}) \backslash \mathbf{pred}_{\mathcal{P}}(X_{\text{in}})$;
  $\mathbf{pa}_{\mathcal{P}'}(X_k) \leftarrow \mathbf{pa}_{\mathcal{P}}(X_{\text{in}})$;
  $\mathbf{pa}_{\mathcal{P}'}(X_{\text{in}}) \leftarrow X_{\text{out}}$;
end

Algorithm 1: Incremental compilation of an arc addition (addArc)

Imagine that we are learning the BN and the PT shown in Fig. 4. Let us focus on some arc additions that cover the three different scenarios. In each example we show the resulting BN after applying the addition of the arc, and the resulting PT after compiling the change in the tree. The obtained PTs are

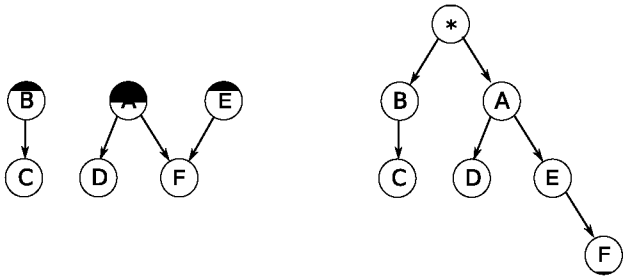


Fig. 4 Examples of BN (*left*) and PT (*right*) respectively

sound but they may be far from optimal, given that we do not include the optimization process in these examples. Figure 5a corresponds to the addition of arc $A \rightarrow E$. In this case A is currently a predecessor of E in the PT (scenario 1), so no changes in the tree are required. Figure 5b corresponds to the addition of $E \rightarrow A$. This change implies a reconfiguration of the network given that A is currently a predecessor of E in the PT (scenario 2), and now E must be set as a predecessor of A without spoiling the soundness of the rest of the nodes. The last example (Fig. 5b) corresponds to the addition of arc $C \rightarrow F$. Given that the only predecessor that C and F have in common is the root node $*$ (scenario 3), C and all its predecessors must be placed as predecessors of F to maintain the soundness of the tree.

3.3.2 Arc removal

The second type of local changes that we need to consider is arc removals. On the one hand, it is straightforward to obtain a sound PT after applying an arc removal in a BN, because it is enough to maintain the current configuration of the tree. On the other hand, a huge reduction in the complexity of the PT may be achieved optimizing the PT after the removal. Algorithm 2 describes the procedure used here for compiling arc removals.

Data: PT $\mathcal{P}$, BN $\mathcal{B}$, output node X_{out}, input node X_{in}
Result: PT $\mathcal{P}'$, BN $\mathcal{B}'$

remove X_{out} from $\mathbf{pa}_{\mathcal{B}'}(X_{\text{in}})$;
$\mathcal{P}' \leftarrow \mathcal{P}$;

Algorithm 2: Incremental compilation of an arc removal (removeArc)

Figure 6 is an example that shows how Algorithm 2 removes arc $A \rightarrow F$ in the BN and PT shown in Fig. 4. There are no changes applied to the PT and it is still sound with respect to the new BN, but a model with a lower inference complexity could be obtained optimizing the tree.

3.3.3 Arc reversal

To compile the reversal of arc $X_{\text{out}} \rightarrow X_{\text{in}}$ (Algorithm 3) we compile first the deletion of arc $X_{\text{out}} \rightarrow X_{\text{in}}$ and then the addition of the reversed arc $X_{\text{in}} \rightarrow X_{\text{out}}$.

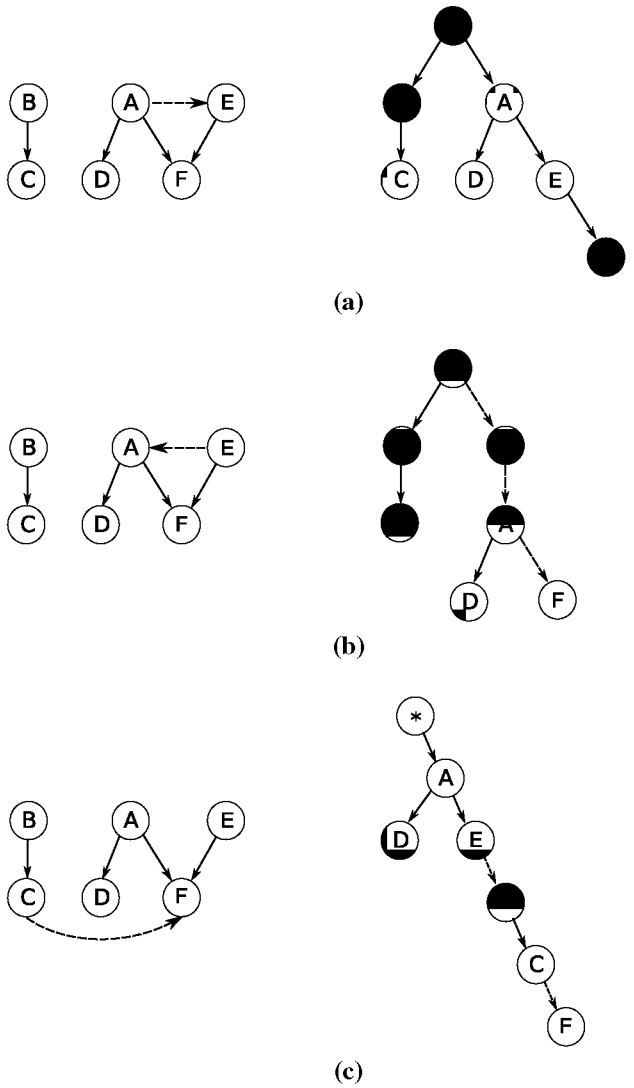


Fig. 5 Example of arc addition for scenarios 1 (**a**), 2 (**b**) and 3 (**c**). BNs on the *left* and PTs on the *right*

Data: PT $\mathcal{P}$, BN $\mathcal{B}$, output node X_{out}, input node X_{in}
Result: PT $\mathcal{P}'$, BN $\mathcal{B}'$
1 $\mathcal{P}_1, \mathcal{B}_1 \leftarrow \text{removeArc}(\mathcal{P}, \mathcal{B}, X_{\text{out}}, X_{\text{in}})$;
2 $\mathcal{P}', \mathcal{B}' \leftarrow \text{addArc}(\mathcal{P}_1, \mathcal{B}_1, X_{\text{out}}, X_{\text{in}})$;

Algorithm 3: Incremental compilation of an arc reversal (reverseArc)

Figure 7 shows an example where we compile the reversal of arc $A \rightarrow F$. The result corresponds to first compiling the removal of arc $A \rightarrow F$ and then compiling the addition of arc $F \rightarrow A$.

3.4 Polynomial tree optimization

Although the methods proposed above assure the soundness of the compiled PTs, the obtained models may have a higher inference complexity than other PTs that are also sound for the same BN. Let us introduce the concept of optimality in PTs:

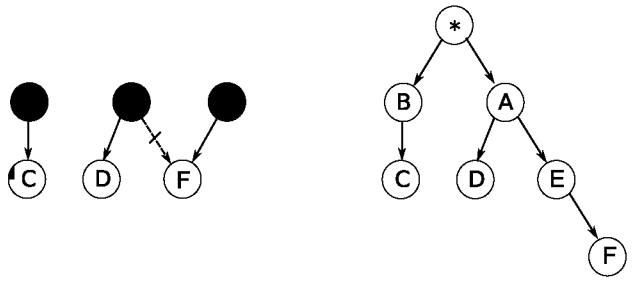


Fig. 6 Example of arc removal. BN (*left*) and PT (*right*)

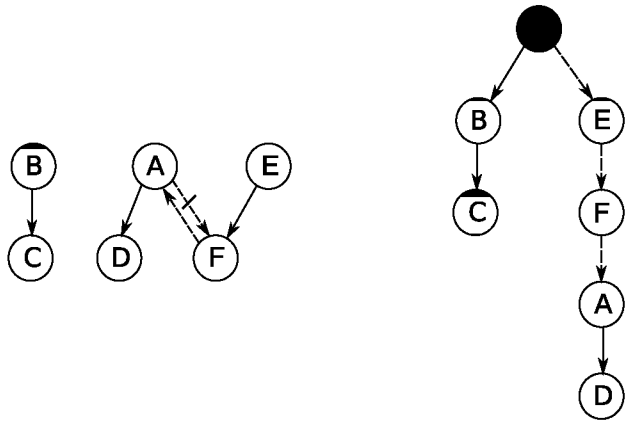


Fig. 7 Example of arc reversal. BN and PT respectively

Definition 2 A PT $\mathcal{P}$ is optimal for a BN $\mathcal{B}$ if $\mathcal{P}$ is sound for $\mathcal{B}$ and there is no other PT $\mathcal{P}'$ such that $\mathcal{P}'$ has a lower inference complexity (measured by Eq. (3)) than $\mathcal{P}$ and is sound for $\mathcal{B}$.

Those PTs that are close to being optimal will represent a tighter bound in the inference complexity of the models. Therefore, our objective is finding PTs that are not only sound, but also close to being optimal.

To avoid the rejection of good solutions because of a poor incremental compilation we perform an optimization process for each PT candidate during the search. The optimization procedure visits iteratively the nodes to be optimized and consists of two phases. The first phase does a smooth optimization, so it visits the deepest node available in the PT in each step. The second phase is only performed if it is possible to reduce the complexity of the PT obtained after the first phase, in which case it visits the shallowest nodes available in the PT to seek bigger changes in the inference complexity.

The key of the optimization process is to find the right local movements that minimize Eq. (3) in each iteration. This task is performed by Algorithm 4, that basically swaps the

position in $\mathcal{P}$ of the node to be optimized X_{opt} with its parent $\mathbf{pa}_{\mathcal{P}}(X_{\text{opt}})$ and checks if the change reduces Eq. (3).

Data: PT $\mathcal{P}$, BN $\mathcal{B}$, node X_{opt}
Result: PT $\mathcal{P}'$

let $\mathcal{P}'$ be a copy of $\mathcal{P}$;
$X_p \leftarrow \mathbf{pa}_{\mathcal{P}}(X_{\text{opt}})$;
if $X_p \in \boldsymbol{pa}_{\mathcal{B}}(X_{\text{opt}})$ **then**
  return ;
end
$\mathbf{pa}_{\mathcal{P}'}(X_{\text{opt}}) \leftarrow \mathbf{pa}_{\mathcal{P}}(X_p)$;
$\mathcal{X}_c \leftarrow (\mathbf{ch}_{\mathcal{P}}(X_{\text{opt}}) \cup \mathbf{ch}_{\mathcal{P}}(X_p)) \backslash \{X_{\text{opt}}\}$;
for $X_i \in \mathcal{X}_c$ **do**
  if $(\{X_i\} \cup \boldsymbol{desc}_{\mathcal{P}}(X_i)) \cap \boldsymbol{desc}_{\mathcal{B}}(X_p) \neq \varnothing$ **then**
    $\mathbf{pa}_{\mathcal{P}'}(X_i) \leftarrow X_p$;
  else
    $\mathbf{pa}_{\mathcal{P}'}(X_i) \leftarrow X_{\text{opt}}$;
  end
end
if $(\{X_p\} \cup \boldsymbol{desc}_{\mathcal{P}'}(X_p)) \cap \boldsymbol{desc}_{\mathcal{B}}(X_{\text{opt}}) \neq \varnothing$ **then**
  $\mathbf{pa}_{\mathcal{P}'}(X_p) \leftarrow X_{\text{opt}}$;
end

Algorithm 4: Optimization step (pushUpNode)

Let $\mathcal{B}$ and $\mathcal{P}$ be the BN and PT received as an input by Algorithm 4 and $\mathcal{P}'$ the PT that it returns. We will sometimes refer to $\mathbf{pa}_{\mathcal{P}}(X_i)$ as P_i and to $\mathbf{pa}_{\mathcal{P}}(P_i)$ as P_p. Let us use the section of the PT shown in Fig. 8a as an example to show the operations performed by Algorithm 4. In the first step of the procedure, all the arcs that join X_i and its parent with the branches hanging from them are deleted. The arc that joins P_i with its parent is also deleted. This step is represented in Fig. 8b.

In the second step of the algorithm (Fig. 8c) the method sets P_p as the new parent of X_i in $\mathcal{P}'$. In the third step (Fig. 8d), the arcs from X_i and P_i to the unassigned branches are added. The branches that contain any node that is a descendant of P_i in $\mathcal{B}$ must now hang from P_i, while the rest of the branches should hang from X_i to reduce the inference complexity of the tree.

The last step consists of assigning the new parent of P_i. If P_i or any of its descendants in $\mathcal{P}'$ is also a descendant of X_i in $\mathcal{B}$ Algorithm 4 sets X_i as the new parent of P_i in $\mathcal{P}'$, as shown in Fig. 8e. Otherwise, P_p continues to be the parent of P_i in $\mathcal{P}'$ (Fig. 8f).

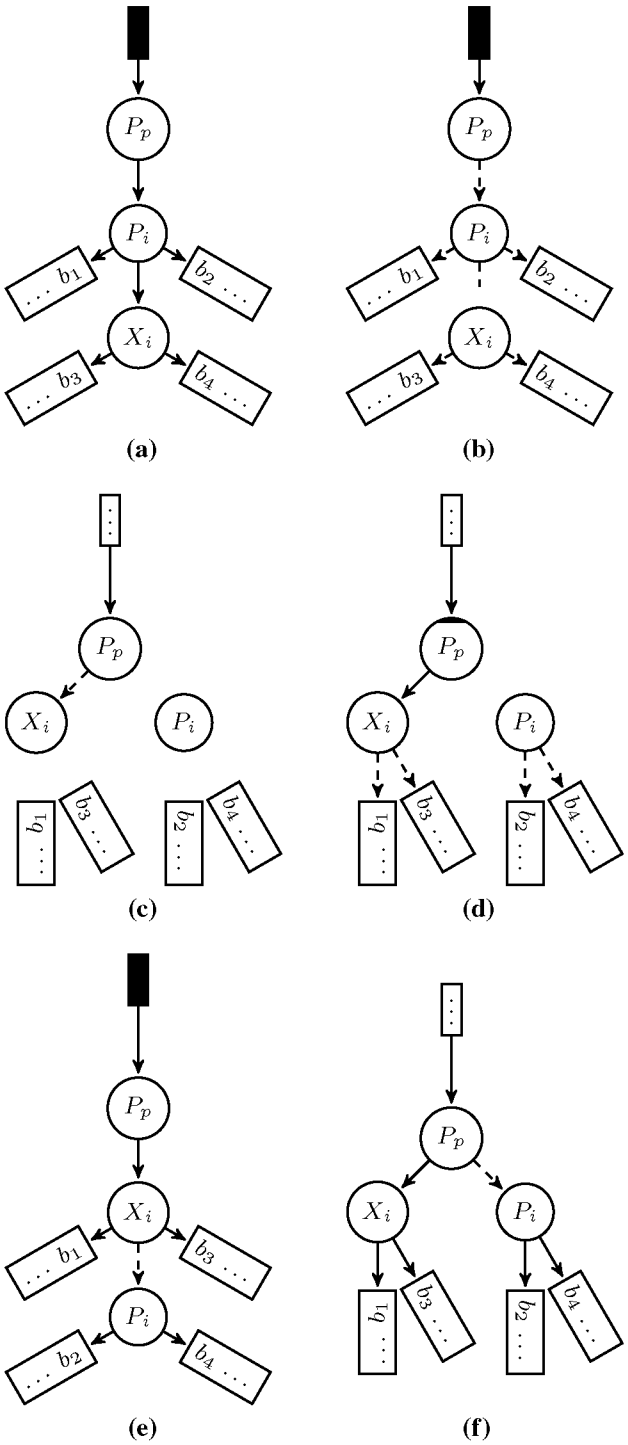


Fig. 8 Example of Algorithm 4

3.5 Learning polynomial trees from data

It is straightforward to learn PTs in combination with any score+search BN learning method that applies local changes during the search. In each step of the learning process we should use the compilation and optimization procedures shown above and then penalize each candidate for its inference complexity, given by Eq. (3).

In this work we penalize the log-likelihood (LL) to measure the accuracy of each model, favoring candidates with

low inference complexity. For a dataset D of size N, the scoring function is defined as:

$$scorePT(\mathcal{B}, \mathcal{P}, D) = LL(\mathcal{B}, D) - k_n \cdot \text{eval}(\mathcal{P}, *) - k_p \cdot |\mathcal{B}| \quad (4)$$

where k_n and k_p represent the weight of inference complexity and of the number of parameters of the BN $|\mathcal{B}|$ respectively for the model penalization.

We need to be sure that the learned PTs are sound, because otherwise they will not be useful. Theorem 1 (proof in "Appendix") assures that any method that uses only the procedures proposed above to make incremental changes in PTs will always obtain sound PTs.

Theorem 1 *Let $\mathcal{P}$ be a sound PT with respect to a BN $\mathcal{B}$, and $\mathcal{A}$ an algorithm that receives $\mathcal{P}$ and $\mathcal{B}$ and obtains a new PT $\mathcal{P}'$ and BN $\mathcal{B}'$. If every change in $\mathcal{P}$ and $\mathcal{B}$ made by $\mathcal{A}$ corresponds to applying Algorithms* 1–4, *then $\mathcal{P}'$ is sound with respect to $\mathcal{B}'$.*

To learn the structure of PTs, we use the methods proposed above in combination with the two iterations constrained hill-climbing (2iCHC) algorithm [19]. 2iCHC is a version of the hill-climbing (HC) algorithm that uses a forbidden parents list to constrain the search space during the learning process, reducing the learning time of HC while assuring the return of a minimal I-map. We call the resulting method hill-climbing for polynomial trees (HCPT).

4 Experimental results

In this section we show and discuss the results obtained for inference and learning using PTs. The idea is to check the impact of including the PT framework to the original method, in this case 2iCHC, and compare the accuracy of inference and the computational cost in both models.

The datasets used in this work were generated from three real-world BNs. WIN95PTS is a medium network for handling printer troubleshooting in Windows 95, PATHFINDER is a large network for the diagnosis of lymph-node diseases [21], and MUNIN1 is a large size network for the diagnosis of neuromuscular disorders [2]. The basic properties of each BN are shown in Tables 1, 2 and 3. We have generated 25,000 learning samples and 40,000 testing samples from each network.

To evaluate the inference accuracy we have used the mean square error (MSE) between the results obtained performing inference in the learned model (Q) and the probability in the test dataset (P). The error is computed using a set of 500 samples from the test data, while the rest of the samples are used to compute the probability of each query in the test dataset (P). From each sample we generate a conditional probability query $P(\boldsymbol{V}|\boldsymbol{E})$ with randomly selected query ($\boldsymbol{V}$) and evidence ($\boldsymbol{E}$) variables. In each test we vary the number of evidence variables from 10 to 25 % of the total, letting the number of query variables fixed at 15 %. The MSE is defined by:

$$\text{MSE}(P, Q) = \frac{1}{m}\sum_{i=1}^{m}(P(\boldsymbol{v}(i)|\boldsymbol{e}(i)) - Q(\boldsymbol{v}(i)|\boldsymbol{e}(i)))^2$$

where $\boldsymbol{v}(i)$ is the instantiation of $\boldsymbol{V}$ in sample i, $\boldsymbol{e}(i)$ is the instantiation of $\boldsymbol{E}$ in sample i, and m is the number of samples.

Table 1 Basic properties of WIN95PTS

Number of nodes	76
Number of arcs	112
Number of parameters	574
Average Markov blanket size	5.92

Table 2 Basic properties of PATHFINDER

Number of nodes	135
Number of arcs	200
Number of parameters	77,155
Average Markov blanket size	3.04

Table 3 Basic properties of MUNIN1

Number of nodes	186
Number of arcs	273
Number of parameters	15,622
Average Markov blanket size	3.81

4.1 Learning results

One of the objectives of this work is to provide a framework that allows computing an upper bound in the inference complexity and that can be easily adapted to most score+search methods. Therefore, we were interested in comparing an existing BN learning method with a modified version of this method using PTs.

We compare the BNs obtained with 2iCHC with the PTs obtained with HCPT, that is a modified version of 2iCHC adapted to learning PTs. We use the minimum description length (MDL) [10] scoring function to evaluate each BN obtained with 2iCHC and Eq. (4) to evaluate each PT.

The results (Tables 4, 5, 6) show that the differences in the likelihood are small. 2iCHC performs better in WIN95PTS and MUNIN1 and HCPT obtains a better result

Table 4 Learning results for WIN95PTS

	2iCHC	HCPT
Log_likelihood	−9.11	−9.62
Number of arcs	120	131
Number of parameters	620	435
Learning time	0 h 12 min	0 h 14 min

Table 5 Learning results for PATHFINDER

	2iCHC	HCPT
Log_likelihood	−27.19	−26.75
Number of arcs	138	140
Number of parameters	1266	1273
Learning time	1 h 51 min	2 h 34 min

Table 6 Learning results for MUNIN1

	2iCHC	HCPT
Log_likelihood	−41.78	−45.73
Number of arcs	220	210
Number of parameters	2085	2190
Learning time	5 h 37 min	6 h 53 min

in PATHFINDER. However, the accuracy of the models is compared in more detail by the inference experiments.

The computational cost of learning PTs with HCPT is a bit higher than the cost of learning BNs with 2iCHC, but the time needed for the incremental compilation of PTs is small compared with the time spent by the scores. Nevertheless, we focus on reducing the inference complexity rather than the learning time, given that the learning process is usually performed only once, while inference is usually performed multiple times.

4.2 Inference results

Next, we compare the performance of exact inference in PTs obtained with HCPT with approximate inference in BNs learned with 2iCHC. We use the likelihood weighting (LW) algorithm for approximate inference [18,29]. The reason for using approximate inference is that the MDL score, that is used in combination with 2iCHC, does not penalize the inference complexity of the models, so the computational cost of performing exact inference in these models is too high.

We use three different sampling sizes for likelihood weighting: quick (200 samples), medium (1000 samples) and slow (2000 samples). This way, we can compare the efficiency and accuracy of the PTs obtained with HCPT with a very fast inference procedure and also with a slower one

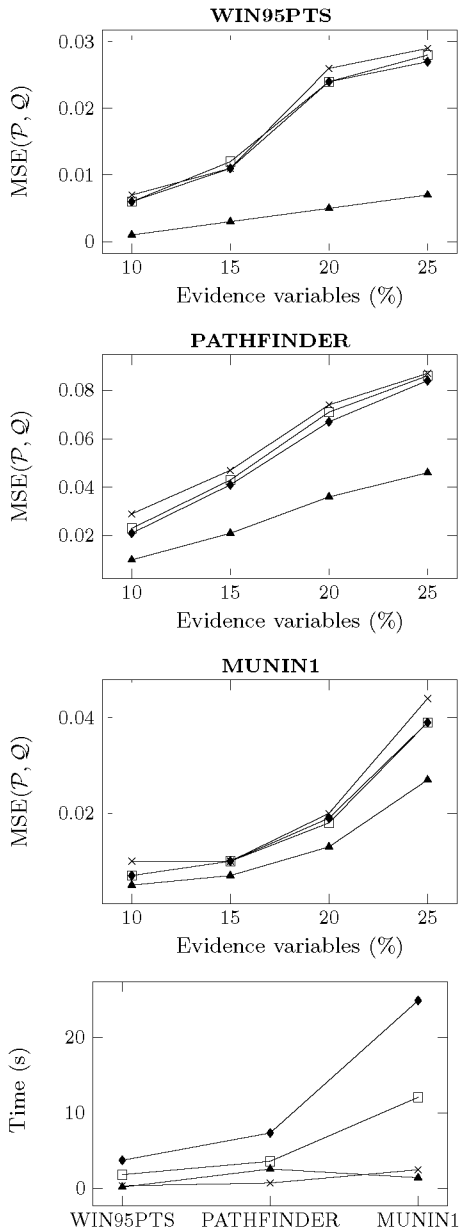


Fig. 9 Inference results comparing quick LW (100 samples) —×—, medium LW (1000 samples) —□—, slow LW (2000 samples) —◆—, and exact inference with PTs —▲—. The computational cost displayed is the mean time (in s) of all the queries answered by each BN

that achieves a better convergence to the target probability distribution. The focus is not on comparing the inference procedures, but on analyzing how penalizing the inference complexity of the models affects the efficiency and accuracy of inference. The results are presented in Fig. 9.

The inference results show that using PTs improves the accuracy of the answers provided by the models obtained with 2iCHC in combination with LW in every scenario. The computational cost of performing exact inference in PTs is always lower than the cost of using 2iCHC and medium (1000 samples) or slow (2000 samples) LW, and similar to the cost

of performing quick LW (200 samples), that produces always the less accurate answers in the tests.

5 Conclusions and future research

We developed a new framework for learning low inference complexity BNs. For that, we have used a simple and intuitive representation (PTs) that allows evaluating the inference complexity of the candidate models during the learning process efficiently. PTs do not exploit the local structures of the network as ACs and they are not as tight as JTs for bounding the complexity of the networks, but they allow a more flexible learning process of low inference complexity models. We have created methods for incrementally compiling and optimizing this representation while learning BNs.

Experimental results show that using the incremental compilation of PTs combined with existing BN learning methods obtains models with low inference complexity requiring a computational cost that is similar to the cost required by the original BN learner. In the tests, the accuracy of the answers provided by exact inference in PTs outperformed those provided by the models learned with a state-of-the-art BN learning method using approximate inference.

Future research will focus on adapting the incremental compilation and optimization methods presented here to representations that are not restricted to topological orders. We are interested in learning tractable models in the space of elimination orders, using an equivalence class of elimination orders and DAGs. This way, we aim to use a representation that provides a tighter upper bound in the inference complexity, such as JTs, but maintaining the flexibility of the representation used in this work.

Most probable explanations can be computed in polynomial time in the number of variables by a class-bridge decomposable multidimensional Bayesian classifier [6] if the number of variables in each of its components and the treewidth of its structure are bounded [23]. We are interested in adapting the methods presented here to learn tractable multidimensional Bayesian classifiers.

Appendix: Proof of Theorem 1

This work relies heavily on Theorem 1, which assures that the proposed incremental compilation and optimization methods produce always sound PTs. To demonstrate the soundness of a PT $\mathcal{P}$ with respect to a BN $\mathcal{B}$, we show that for each node X_i of $\mathcal{P}$ every parent of X_i in $\mathcal{B}$ is a predecessor of X_i in $\mathcal{P}$. In this "Appendix" we provide a proof of Theorem 1.

Lemma 1 *Let $\mathcal{P}$ be a PT over $\mathcal{X}_P = \{*\} \cup \mathcal{X}$ and $\mathcal{B}$ be a Bayesian network over $\mathcal{X}$. If $\mathcal{P}$ is sound with respect to $\mathcal{B}$, then the PT $\mathcal{P}'$ obtained after applying addArc($\mathcal{B}$, $\mathcal{P}$, X_{out}, X_{out}) is also sound with respect to $\mathcal{B}'$, where $\mathcal{B}'$ is the result of adding arc $X_{\text{out}} \rightarrow X_{\text{in}}$ to $\mathcal{B}$, and the addition of $X_{\text{out}} \rightarrow X_{\text{in}}$ to $\mathcal{B}$ does not produce a cycle in $\mathcal{B}'$.*

Proof The structure of $\mathcal{P}'$ depends on the precedence relationship between X_{out} and X_{in} in $\mathcal{P}$.

- $X_{\text{out}} \in \mathbf{pred}_{\mathcal{P}}(X_{\text{in}})$: there are no changes in the structure of $\mathcal{P}$. $\forall X_i \in \mathcal{X}\backslash\{X_{\text{in}}\}$, $\mathbf{pa}_{\mathcal{B}'}(X_i) = \mathbf{pa}_{\mathcal{B}}(X_i)$ and $\mathbf{pred}_{\mathcal{P}'}(X_i) = \mathbf{pred}_{\mathcal{P}}(X_i)$, so X_i is sound. X_{in} is also sound because $\mathbf{pa}_{\mathcal{B}'}(X_{\text{in}}) = \mathbf{pa}_{\mathcal{B}}(X_{\text{in}}) \cup \{X_{\text{out}}\}$, $\mathbf{pred}_{\mathcal{P}'}(X_{\text{in}}) = \mathbf{pred}_{\mathcal{P}}(X_{\text{in}})$ and $X_{\text{out}} \in \mathbf{pred}_{\mathcal{P}}(X_{\text{in}})$.
- $X_{\text{in}} \in \mathbf{pred}_{\mathcal{P}}(X_{\text{out}})$: The nodes that are not descendants of X_{in} in $\mathcal{P}$ do not change. $\forall X_i \in \mathcal{X}\backslash(\mathbf{desc}_{\mathcal{P}}(X_{\text{in}}) \cup \{X_{\text{in}}\})$, $\mathbf{pa}_{\mathcal{B}'}(X_i) = \mathbf{pa}_{\mathcal{B}}(X_i)$ and $\mathbf{pred}_{\mathcal{P}'}(X_i) = \mathbf{pred}_{\mathcal{P}}(X_i)$. Thus, X_i is sound. X_{out} and its descendants in $\mathcal{P}'$ that are not descendants of X_{in} have less predecessors in $\mathcal{P}'$ than in $\mathcal{P}$. $\forall X_i \in \mathbf{desc}_{\mathcal{P}'}(X_{\text{out}}) \cup \{X_{\text{out}}\}\backslash(\mathbf{desc}_{\mathcal{P}'}(X_{\text{in}}) \cup \{X_{\text{in}}\})$, as $\mathbf{pred}_{\mathcal{P}'}(X_i) = \mathbf{pred}_{\mathcal{P}}(X_i)\backslash(\mathbf{desc}_{\mathcal{P}'}(X_{\text{in}})\cup\{X_{\text{in}}\})$, $\mathbf{pa}_{\mathcal{B}'}(X_i) = \mathbf{pa}_{\mathcal{B}}(X_i)$ and $\mathbf{pa}_{\mathcal{B}'}(X_i) \cap (\mathbf{desc}_{\mathcal{P}'}(X_{\text{in}}) \cup \{X_{\text{in}}\}) = \varnothing$, X_i is sound. Finally, X_{in} has X_{out} as a predecessor in $\mathcal{P}$. $\forall X_i \in \mathbf{desc}_{\mathcal{P}'}(X_{\text{in}}) \cup \{X_{\text{in}}\}$, $\mathbf{pred}_{\mathcal{P}'}(X_i) \supseteq \mathbf{pred}_{\mathcal{P}}(X_i) \cup \{X_{\text{out}}\}$ and $\mathbf{pa}_{\mathcal{B}'}(X_i) \subseteq \mathbf{pa}_{\mathcal{B}}(X_i) \cup \{X_{\text{out}}\}$, so X_i is sound.
- $X_{\text{out}} \notin \mathbf{pred}_{\mathcal{P}}(X_{\text{in}})$ and $X_{\text{in}} \notin \mathbf{pred}_{\mathcal{P}}(X_{\text{out}})$: X_{out} and its predecessors in $\mathcal{P}$ are set as predecessors of X_{in} in $\mathcal{P}'$. $\forall X_i \notin \mathbf{desc}_{\mathcal{P}'}(X_{\text{in}}) \cup \{X_{\text{in}}\}$, $\mathbf{pa}_{\mathcal{B}'}(X_i) = \mathbf{pa}_{\mathcal{B}}(X_i)$ and $\mathbf{pred}_{\mathcal{P}'}(X_i) \supseteq \mathbf{pred}_{\mathcal{P}}(X_i)$. Hence X_i is sound. $\forall X_i \in \mathbf{desc}_{\mathcal{P}'}(X_{\text{in}}) \cup \{X_{\text{in}}\}$, $\mathbf{pa}_{\mathcal{B}'}(X_i) \subseteq \mathbf{pa}_{\mathcal{B}}(X_i) \cup \{X_{\text{out}}\}$ and $\mathbf{pred}_{\mathcal{P}'}(X_i) \supseteq \mathbf{pred}_{\mathcal{P}}(X_i) \cup \{X_{\text{out}}\}$. Therefore X_i is sound.

Lemma 2 *Let $\mathcal{P}$ be a PT over $\mathcal{X}_P = \{*\} \cup \mathcal{X}$ and $\mathcal{B}$ be a Bayesian network over $\mathcal{X}$. If $\mathcal{P}$ is sound with respect to $\mathcal{B}$, then the PT $\mathcal{P}'$ obtained after applying removeArc($\mathcal{B}$, $\mathcal{P}$, X_{out}, X_{in}) is also sound with respect to $\mathcal{B}'$, where $\mathcal{B}'$ is the result of removing arc $X_{\text{out}} \rightarrow X_{\text{in}}$ from $\mathcal{B}$.*

Proof $\forall X_i \in \mathcal{X}$, $\mathbf{pa}_{\mathcal{B}'}(X_i) \subseteq \mathbf{pa}_{\mathcal{B}}(X_i)$ and $\mathbf{pred}_{\mathcal{P}'}(X_i) = \mathbf{pred}_{\mathcal{P}}(X_i)$, so X_i is sound.

Lemma 3 *Let $\mathcal{P}$ be a PT over $\mathcal{X}_P = \{*\} \cup \mathcal{X}$ and $\mathcal{B}$ be a Bayesian network over $\mathcal{X}$. If $\mathcal{P}$ is sound with respect to $\mathcal{B}$, then the PT $\mathcal{P}'$ obtained after applying reverseArc($\mathcal{B}$, $\mathcal{P}$, X_{out}, X_{in}) is also sound with respect to $\mathcal{B}'$, where $\mathcal{B}'$ is the result of reversing arc $X_{\text{out}} \rightarrow X_{\text{in}}$ in $\mathcal{B}$, and $X_{\text{in}} \rightarrow X_{\text{out}}$ does not produce a cycle in $\mathcal{B}'$.*

Proof We can describe the reversion of arc $X_{\text{out}} \rightarrow X_{\text{in}}$ in two steps:

1 $\mathcal{P}_1, \mathcal{B}_1 \leftarrow$ removeArc($\mathcal{P}$, $\mathcal{B}$, X_{out}, X_{in}).

2. $\mathcal{P}', \mathcal{B}' \leftarrow \text{addArc}(\mathcal{P}_1, \mathcal{B}_1, X_{\text{in}}, X_{\text{out}})$.

From Lemma 1 we know that $\mathcal{P}_1$ is sound with respect to $\mathcal{B}_1$, and from Lemma 2 we know that $\mathcal{P}'$ is sound with respect to $\mathcal{B}'$.

Lemma 4 *Let $\mathcal{P}$ be a PT over $\mathcal{X}_P = \{*\} \cup \mathcal{X}$ and $\mathcal{B}$ be a Bayesian network over $\mathcal{X}$. If $\mathcal{P}$ is sound with respect to $\mathcal{B}$, then the PT $\mathcal{P}'$ obtained after applying pushUpNode($\mathcal{B}$, $\mathcal{P}$, X_{opt}) is also sound with respect to $\mathcal{B}$.*

Proof Let $\mathcal{D}_{\text{opt}} = (\mathbf{desc}_{\mathcal{P}'}(X_{\text{opt}}) \cup \{X_{\text{opt}}\}) \backslash (\mathbf{desc}_{\mathcal{P}'}(X_p) \cup \{X_p\})$, and $\mathcal{D}_p = \mathbf{desc}_{\mathcal{P}'}(X_p) \cup \{X_p\}$.

$\forall X_i \in \mathcal{X} \backslash (\mathcal{D}_{\text{opt}} \cup \mathcal{D}_p)$, $\mathbf{pred}_{\mathcal{P}'}(X_i) = \mathbf{pred}_{\mathcal{P}}(X_i)$. Therefore, X_i is sound.

$\forall X_i \in \mathcal{D}_{\text{opt}}$, $\mathbf{pred}_{\mathcal{P}'}(X_i) = \mathbf{pred}_{\mathcal{P}}(X_i) \backslash \{X_p\}$. Given that $X_i \in \mathcal{D}_{\text{opt}}$ only if $X_i \notin \mathbf{desc}_{\mathcal{B}}(X_p)$, then X_i is sound.

The nodes in $\mathcal{D}_p$ may contain X_{opt} as a predecessor in $\mathcal{P}'$ depending on their predecessors in $\mathcal{B}$.

If $\mathcal{D}_p \cap \mathbf{desc}_{\mathcal{B}}(X_{\text{opt}}) \neq \varnothing$, $\forall X_i \in \mathcal{D}_p$, $\mathbf{pred}_{\mathcal{P}'}(X_i) = \mathbf{pred}_{\mathcal{P}}(X_i) \cup \{X_{\text{opt}}\}$, so X_i is sound. Otherwise, $\forall X_i \in \mathcal{D}_p$, $\mathbf{pred}_{\mathcal{P}'}(X_i) = \mathbf{pred}_{\mathcal{P}}(X_i) \backslash \{X_{\text{opt}}\}$, and given that $X_{\text{opt}} \notin \mathbf{pred}_{\mathcal{B}}(X_i)$, X_i is sound.

Theorem 1 *Let $\mathcal{P}$ be a sound PT with respect to a BN $\mathcal{B}$, and $\mathcal{A}$ an algorithm that receives $\mathcal{P}$ and $\mathcal{B}$ and obtains a new PT $\mathcal{P}'$ and BN $\mathcal{B}'$. If every change in $\mathcal{P}$ and $\mathcal{B}$ made by $\mathcal{A}$ corresponds to applying Algorithms* 1–4, *then $\mathcal{P}'$ is sound with respect to $\mathcal{B}'$.*

Proof Algorithm $\mathcal{A}$ obtains $\mathcal{P}'$ and $\mathcal{B}'$ from $\mathcal{P}$ and $\mathcal{B}$ using any sequence of changes, where each change is produced by Algorithms 1–4. Since $\mathcal{P}$ is sound for $\mathcal{B}$ and Lemmas 1–4 assure that Algorithms 1–4 return a PT $\mathcal{P}_1$ and a BN $\mathcal{B}_1$ such that $\mathcal{P}_1$ is sound for $\mathcal{B}_1$, the result of applying the sequence of changes in $\mathcal{A}$ is a PT $\mathcal{P}'$ and a BN $\mathcal{B}'$ where $\mathcal{P}'$ is sound for $\mathcal{B}'$.

References

1. Akaike, H.: A new look at the statistical model identification. IEEE Trans. Autom. Control **19**(6), 716–723 (1974)
2. Andreassen, S., Rosenfalck, A., Falck, B., Olesen, K.G., Andersen, S.K.: Evaluation of the diagnostic performance of the expert EMG assistant MUNIN. Electromyogr. Mot. Control **101**(2), 129–144 (1996)
3. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebraic Discret. **8**(2), 277–284 (1987)
4. Bach, F.R., Jordan, M.I.: Thin junction trees. In: Adv. Neural Inf., pp. 569–576 (2001)
5. Beygelzimer, A., Rish, I.: Approximability of probability distributions. In: Adv. Neural Inf. pp. 377–384 (2004)
6. Bielza, C., Li, G., Larranaga, P.: Multi-dimensional classification with Bayesian networks. Int. J. Approx. Reason. **52**(6), 705–727 (2011)
7. Bielza, C., Larranaga, P.: Discrete Bayesian network classifiers: a survey. ACM Comput. Surv. **47**(1), 5 (2014)
8. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, pp. 226–234 (1993)
9. Bodlaender, H.L., Koster, A.M.: Treewidth computations I. Upper bounds. Inf. Comput. **208**(3), 259–275 (2010)
10. Bouckaert, R.R.: Probabilistic network construction using the minimum description length principle. In: Lect. Notes Artif. Int., pp. 41–48 (1993)
11. Chechetka, A., Guestrin, C.: Efficient principled learning of thin junction trees. In: Adv. Neural Inf., pp. 273–280 (2008)
12. Cooper, G.F.: The computational complexity of probabilistic inference using Bayesian belief networks. Artif. Intell. **42**(2), 393–405 (1990)
13. Cooper, G.F., Herskovits, E.: A Bayesian method for constructing Bayesian belief networks from databases. In: Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence, pp. 86–94 (1991)
14. Cooper, G.F., Herskovits, E.: A Bayesian method for the induction of probabilistic networks from data. Mach. Learn. **9**(4), 309–347 (1992)
15. Dagum, P., Luby, M.: Approximating probabilistic inference in Bayesian belief networks is NP-hard. Artif. Intell. **60**(1), 141–153 (1993)
16. Darwiche, A.: A differential approach to inference in Bayesian networks. J. Assoc. Comput. Mach. **50**(3), 280–305 (2003)
17. Elidan, G., Gould, S.: Learning bounded treewidth Bayesian networks. In: Adv. Neural Inf., pp. 417–424 (2009)
18. Fung, R.M., Chang, K.C.: Weighing and integrating evidence for stochastic simulation in Bayesian networks. In: Uncertainty in Artificial Intelligence, pp. 209–220 (1989)
19. Gámez, J.A., Mateo, J.L., Puerta, J.M.: Learning Bayesian networks by hill climbing: effficient methods based on progressive restriction of the neighborhood. Data Min. Knowl. Discov. **22**(1–2), 106–148 (2011)
20. Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian networks: the combination of knowledge and statistical data. Mach. Learn. **20**(3), 197–243 (1995)
21. Heckerman, D., Horwitz, E., Nathwani, B.: Towards normative expert systems: part I. The pathfinder project. Methods Inf. Med. **31**, 90–105 (1992)
22. Kim, J., Pearl, J.: A computational model for causal and diagnostic reasoning in inference systems. In: Proceedings of the Eighth International Joint Conference on Artificial Intelligence, pp. 190–193 (1983)
23. Kwisthout, J.: Most probable explanations in Bayesian networks: complexity and tractability. Int. J. Approx. Reason. **52**(9), 1452–1469 (2011)
24. Lam, W., Bacchus, F.: Learning Bayesian belief networks: an approach based on the MDL principle. Comput. Intell. **10**(3), 269–293 (1994)
25. Larranaga, P., Kuijpers, C.M., Murga, R.H., Yurramendi, Y.: Learning Bayesian network structures by searching for the best ordering with genetic algorithms. IEEE Trans. Syst. Man Cybern. **26**(4), 487–493 (1996)
26. Larrañaga, P., Poza, M., Yurramendi, Y., Murga, R.H., Kuijpers, C.M.: Structure learning of Bayesian networks by genetic algorithms: a performance analysis of control parameters. IEEE Trans. Pattern Anal. **18**(9), 912–926 (1996)
27. Lowd, D., Domingos, P.: Learning arithmetic circuits. In: Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence, pp. 383–392 (2008)

28. Pham, D.T., Ruz, G.A.: Unsupervised training of Bayesian networks for data clustering. Proc. Roy. Soc. Lond. A Mat., pp. 2927–2948 (2009)
29. Shachter, R.D., Peot, M.A.: Simulation approaches to general probabilistic inference on belief networks. In: Uncertainty in Artificial Intelligence, pp. 221–234 (1989)
30. Shahaf, D., Guestrin, C.: Learning thin junction trees via graph cuts. In: International Conference on Artificial Intelligence and Statistics, pp. 113–120 (2009)
31. Schwarz, G.: Estimating the dimension of a model. Ann. Stat. **6**(2), 461–464 (1978)