Artificial Intelligence 274 (2019) 66–90

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Learning tractable Bayesian networks in the space of elimination orders

Marco Benjumeda *, Concha Bielza, Pedro Larrañaga

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

ARTICLE INFO

Article history:
Received 10 January 2018
Received in revised form 18 August 2018
Accepted 19 November 2018
Available online 12 February 2019

Keywords:
Learning tractable models
Inference complexity
Bayesian networks
Treewidth estimation
Machine learning

ABSTRACT

The computational complexity of inference is now one of the most relevant topics in the field of Bayesian networks. Although the literature contains approaches that learn Bayesian networks from high dimensional datasets, traditional methods do not bound the inference complexity of the learned models, often producing models where exact inference is intractable. This paper focuses on learning tractable Bayesian networks from data. To address this problem, we propose strategies for learning Bayesian networks in the space of elimination orders. In this manner, we can efficiently bound the inference complexity of the networks during the learning process. Searching in the combined space of directed acyclic graphs and elimination orders can be extremely computationally demanding. We demonstrate that one type of elimination trees, which we define as valid, can be used as an equivalence class of directed acyclic graphs and elimination orders, removing redundancy. We propose methods for incrementally compiling local changes made to directed acyclic graphs in elimination trees and for searching for elimination trees of low width. Using these methods, we can move through the space of valid elimination trees in polynomial time with respect to the number of network variables and in linear time with respect to treewidth. Experimental results show that our approach successfully bounds the inference complexity of the learned models, while it is competitive with other state-of-the-art methods in terms of fitting to data.

1. Introduction

Bayesian networks (BNs) [1,2] concisely model probability distributions over a set of random variables. They are self-explanatory and easy to understand, and they are well suited for representing causal relationships. Some applications of BNs are supervised classification [3] and clustering [4,5]. Each BN $\mathcal{B}$ over a set of variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ is composed of:

- A *directed acyclic graph* (DAG) $\mathcal{G}$ that represents the conditional independences among triplets of variables in $\mathcal{X}$.
- A set of *parameters* $P(X_i|\mathbf{Pa}_{\mathcal{G}}(X_i))$ that represent the conditional probability distributions of each $X_i \in \mathcal{X}$ conditional on its parents **Pa** in $\mathcal{G}$.

* Corresponding author.
E-mail addresses: marco.benjumeda.barquita@upm.es (M. Benjumeda), mcbielza@fi.upm.es (C. Bielza), pedro.larranaga@fi.upm.es (P. Larrañaga).

https://doi.org/10.1016/j.artint.2018.11.007
0004-3702/

A common approach for learning BNs from data is to perform a search process optimizing a scoring function that measures the quality of each structure. Two types of scores are usually used. Bayesian metrics maximize the posterior probability of the network conditional on the data given a prior distribution over all the possible network structures, while information theory metrics try to maximize the data compression achieved by each network. Well-known scoring functions such as Bayesian Dirichlet equivalent (BDe) [6], K2 [7,8], Akaike information criterion (AIC) [9], Bayesian information criterion (BIC) [10] or minimum description length (MDL) [11,12] implicitly or explicitly penalize the number of network parameters. Note that BIC, that is based on the Schwarz Information Criterion [10], is equivalent to MDL as a BN scoring function. The representation complexity, which is given by the number of network parameters, does not place an upper bound on the inference complexity of the models, and a model with a low representation complexity can have a high inference complexity. Thus, more precise estimations of the inference complexity are required to ensure the tractability of models during the learning process.

A good indicator of the inference complexity of a BN $\mathcal{B}$ with structure $\mathcal{G}$ is the treewidth of $\mathcal{G}$ ($\text{tw}(\mathcal{G})$), given that the most widely used exact inference methods for BNs, like variable elimination (VE) or message passing in junction trees (JTs), can be computed in exponential time in $\text{tw}(\mathcal{G})$. Intuitively, the treewidth of a graph $\mathcal{G}$ can be understood as a measure of how similar $\mathcal{G}$ is to a tree (e.g., a tree has treewidth one). It is NP-hard to determine the treewidth of a graph [13], and there are no efficient exact methods for solving this problem. Many heuristics have been proposed for treewidth estimation (see Section 2.2), but most are very computationally demanding. This is especially important when BNs have to be learned from data, since we have to compute the treewidth of each candidate during the learning process to ensure tractability.

VE is one of the simplest methods for inference in BNs. It consists of successively eliminating the variables of a network until it yield the answer to a given query. The elimination of a variable X_i consists of outputting the product of all the factors containing X_i, and marginalizing the result over X_i. The order in which the variables are removed is called elimination order (EO). The computational cost of VE is upper bounded by the width of the chosen EO π, which is the number of variables in the biggest factor induced by VE minus one.

The treewidth of a graph $\mathcal{G}$ can also be expressed as the width of the optimal EO π_{opt} for graph $\mathcal{G}$. This means that obtaining an optimal EO of $\mathcal{G}$ is equivalent to obtaining the treewidth of $\mathcal{G}$ [13], and it is also an NP-hard problem. Hence, one way of getting an accurate estimation of $\text{tw}(\mathcal{G})$ is to find a good EO for $\mathcal{G}$. It would often be intractable to get a good EO from scratch for each candidate network during the structure search. As most structure learning methods perform local changes in $\mathcal{G}$ during the learning process, a more efficient solution to this problem is to incrementally update the EOs for each local change performed in $\mathcal{G}$. There are usually multiple equivalent EOs for $\mathcal{G}$ (see Section 2.1). This means that the combined space of DAGs and EOs is highly redundant, and it would be extremely computationally demanding to search for low complexity structures in this space. In this paper, we define a type of elimination trees (ETs) [14], which we call valid ETs, that avoid this redundancy. A single valid ET can be used to represent all the EOs that are equivalent (i.e., induce the same factors during VE) for any graph $\mathcal{G}$. We propose methods for efficiently compiling each possible local change that could be applied in $\mathcal{G}$ (i.e., arc additions, removals or reversals), and provide a framework for learning valid ETs from data using the above methods.

This paper is organized as follows. Section 2 introduces inference complexity in BNs and reviews previous work on bounding the treewidth of BNs and learning models of low inference complexity. Section 3 contains our proposal. We show the relation between ETs and EOs, and the way the former can be used as an equivalence class of EOs and DAGs. Section 4 describes the proposed compilation and optimization methods, and it shows how to use ETs to learn tractable BNs in the space of EOs. Section 5 reports the experimental results. Section 6 outlines the concluding remarks and future research lines.

The software of the proposed method is available at https://github.com/marcobb8/et-learn.

2. Background

2.1. Treewidth and elimination orders

To give a formal definition of treewidth, we must first introduce moral graphs.

Definition 1. (Moral graph) The moral graph $\mathcal{G}_M$ of a directed graph $\mathcal{G}$ with nodes $\mathcal{X}$ is the result of:

1. Adding an undirected link between each pair of nodes $X_i, X_j \in \mathcal{X}$ that have a common child in $\mathcal{G}$ and are not connected.
2. Converting every directed arc into an undirected link.

Next, we define the *tree decomposition* of a graph, also known as *jointree* or *junction tree*.

Definition 2. (Tree decomposition graph) Let $\mathcal{G}_M$ be the moral graph of a directed graph $\mathcal{G}$ with nodes $\mathcal{X}$. A tree decomposition of $\mathcal{G}$ is a tree $\mathcal{T}$ with a set of clusters $\mathcal{C}$, where each cluster $C_i \in \mathcal{C}$ is a node of $\mathcal{T}$, that satisfy:

- Each cluster C_i of $\mathcal{T}$ is a subset of $\mathcal{X}$.

- For all edges $X_i - X_j$ in $\mathcal{G}_M$, there is a cluster $C_k \in \mathcal{C}$ such that $X_i, X_j \in C_k$.
- If a node X_i appears in two clusters C_i and C_j, it must also appear in every cluster C_k on the path connecting C_i and C_j in the tree decomposition (the running intersection property).

The width of a tree decomposition $\mathcal{T}$ is $\max_{C_i \in \mathcal{C}} |C_i| - 1$ (i.e., the size of its biggest cluster minus 1). The treewidth of a graph $\mathcal{G}$ is the minimum width across all its decompositions.

An EO of a set of variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ is a permutation $\pi = (\pi(X_1), \ldots, \pi(X_n))$ of $\mathcal{X}$. We use $(X_i < X_j)_\pi$ to denote that X_i must be eliminated before X_j given π.

The treewidth of a graph $\mathcal{G}$ can also be expressed as the width of its optimal EO. As it is NP-hard to find an optimal EO, several heuristics are used to find EOs that are satisfiable in practice. In Section 2.2 we discuss some options.

The combined space of EOs and DAGs is redundant. This means that there may be multiple EOs that induce the same factors (using VE) for the same BN. We define the equivalence of two EOs as:

Definition 3. (Equivalence of EOs) Let $\mathcal{B}$ be a BN over $\mathcal{X}$, and π_1 and π_2 two EOs of $\mathcal{X}$. Let $\mathbf{Cls}_{\pi_1}(X_i)$ and $\mathbf{Cls}_{\pi_2}(X_i)$ be the clusters induced by visiting node X_i during VE using the EOs π_1 and π_2, respectively. π_1 and π_2 are equivalent for $\mathcal{B}$ if, for each $X_i \in \mathcal{X}$, $\mathbf{Cls}_{\pi_1}(X_i) = \mathbf{Cls}_{\pi_2}(X_i)$.

The completeness of a set of EOs $\mathcal{S}$ for $\mathcal{B}$ ensures that if an π_i belongs to $\mathcal{S}$ all the EOs that are equivalent to π_i for $\mathcal{B}$ also belong to $\mathcal{S}$. Note that the completeness of $\mathcal{S}$ does not imply that all the nodes in $\mathcal{S}$ are equivalent for $\mathcal{B}$.

Definition 4. (Completeness of a set of EOs) A set of EOs $\mathcal{S}$ is complete for $\mathcal{B}$ if there are no two equivalent EOs π_i, π_j, with $\pi_i \in \mathcal{S}$ and $\pi_j \notin \mathcal{S}$, for $\mathcal{B}$.

For example, assume a network $\mathcal{B}$ over variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ that represents the product of marginals $P(X_1, \ldots, X_n) = P(X_1)P(X_2)\cdots P(X_n)$. Given $\mathcal{B}$, VE induces the same factors for any EO of $X_1, \ldots, X_n$. Hence, all the $n!$ possible EOs are equivalent for $\mathcal{B}$, and there is a single complete set of EOs that contains all the permutations of $\mathcal{X}$.

2.2. Treewidth estimation

It is NP-hard to exactly compute the treewidth of a BN [13]. There are many approaches whose time complexity is exponential in the number of network variables [15–18]. In practice, heuristics are most often used. As the treewidth of a graph $\mathcal{G}$ is given by the width of its optimal EO, some well-known heuristics estimate the treewidth of $\mathcal{G}$ by searching for good EOs for $\mathcal{G}$. The list below includes some popular approaches:

- Greedy search methods: Two widely used strategies are to eliminate, at each iteration, the smallest degree node (i.e., the node with fewest neighbors) in the graph [19] or the node that produces the minimum number of fill-in (min-fill) edges [20]. In practice, the min-fill algorithm performance is generally slightly better, but its computational cost is higher.
- Graph recognition techniques: The lexicographic breadth-first search algorithm (LEX) [21] and the maximum cardinality search (MCS) algorithm [22] return an optimal EO only if the input graph is chordal.[1] The chordality assumption is very restrictive in practice, but there are two variants of these methods, respectively called LEX-M [21] and MCS-M [23], that also search for a good EO if the graph is not chordal.
- Local search and evolutionary techniques: Some well-known heuristics like simulated annealing [24], genetic algorithms [25], or tabu search [26] have been used to find good EOs.

Another approach focuses on finding the best graph separators, recursively splitting the clusters of an initial tree decomposition into smaller components [27]. Most methods using this strategy give theoretical guarantees of the treewidth upper bound. Bodlaender and Koster [28] provide an overview of the different heuristics used for computing upper bounds for graph treewidth, including the above methods. Their experiments suggest that greedy search methods outperform graph recognition techniques and approaches that use separators.

Sometimes it is sufficient to check that the treewidth does not exceed a constant k rather than exactly computing the treewidth of $\mathcal{G}$; for instance by learning models with a treewidth less than or equal to k (see Section 2.4). Although this is an NP-complete problem [13], it can be computed in linear time in the number of variables for a fixed k. Nevertheless, the time complexity for solving this inequality is super-exponential in the treewidth of $\mathcal{G}$ [29], which means that it may be intractable unless k is very small.

[1] A graph is chordal if all cycles of four or more nodes have an edge that connects two nodes of the cycle but is not part of the cycle.

2.3. Inference complexity in Bayesian networks

Probabilistic inference can be used to refer to multiple problems in BNs. Some well-known inference problems are: evidence propagation, finding the maximum a posteriori (MAP) hypothesis, and computing the most probable explanation (MPE). Evidence propagation entails finding the posterior probability $P(\boldsymbol{Q}|\boldsymbol{e})$ of a set of query variables $\boldsymbol{Q}$ conditional on evidence $\boldsymbol{e}$. It can be used for some key tasks such as prediction and diagnosis. Finding the MAP consists of searching the most probable configuration of a set of variables in a BN for given evidence. The MPE is a special case of the MAP that involves searching the most probable configuration of all variables not instantiated in a BN for given evidence. Kwisthout [30] provides a thorough overview of the complexity of many MPE and MAP variants.

In this paper, we use inference complexity to refer to the complexity of evidence propagation in BNs. Exact inference in BNs is generally NP-hard [31], and approximate inference is commonly used when exact inference is intractable. Approximate inference in BNs is also NP-hard [32], and, although it has been useful for solving some otherwise intractable problems, it has some major drawbacks. It degrades the responses output by the model, and hardly any of these algorithms offer any guarantees of convergence.

The message-passing (MP) algorithm [33,34] can perform exact linear time inference in the number of variables of any BN $\mathcal{B}$ when its topology is a polytree. However, there are many situations where polytrees are not representative enough, and this restriction is therefore too strict in practice. Inference in BNs with loops is far from straightforward, and we cannot use MP to perform exact inference in this type of networks. Although MP has been adapted to deal with loops, the resultant method, called loopy belief propagation [1], provides only approximate results. Most exact inference methods for graphs with loops are based on variable elimination [35,36], conditioning [37,38] and clustering [39,40]. For any BN $\mathcal{B}$, the above methods are exponential in the treewidth of $\mathcal{B}$. Thus, $\mathrm{tw}(\mathcal{B})$ is a good estimator of the inference complexity of $\mathcal{B}$.

The literature also includes approaches that are not always exponential in the treewidth. In this case, tractable exact inference does not necessarily call for models with a low treewidth. These methods exploit the local network structures [41,42], or the exchangeability between the model variables [43]. Nevertheless, it is extremely challenging to consider the above properties during the learning process.

2.4. Previous work on learning low inference complexity Bayesian networks

Most approaches that address the problem of inference complexity during the learning process use a bound k on the model treewidth (i.e., bounded treewidth models). They reject any candidate $\mathcal{G}$ for which $\mathrm{tw}(\mathcal{G}) > k$. Learning bounded treewidth BNs is an NP-hard problem [44]. The literature contains exact methods for this problem that reduce the problem to either a weighted maximum satisfiability problem [45] or mixed-integer linear programming formulations [46,47]. These methods scale poorly with respect to the number of model variables and model treewidth.

Elidan and Gould [48] proposed a method that uses an incremental triangulation of BNs during the structure search to output bounded treewidth models. Their method is treewidth-friendly (i.e., each update of the triangulation does not increment its width by more than one), and it basically applies the best chain of arc additions in each iteration given a topological ordering of the variables. Its main limitation is that the method is restricted to a single topological ordering of the variables in each iteration.

Nie et al. [46] proposed an efficient approach that samples k-trees randomly and selects the best BN structure whose moral graph is the sampled k-tree. As the convergence of the sampling process can be a problem when the number of variables is not small, Nie et al. [49] also provided a strategy for moving in the space of k-trees and proposed a score (I-score) to measure how well a k-tree fits the data. The authors showed that this measure is correlated with the BDeu score of the learned networks.

Scanagatta et al. [50] proposed a method (called k-greedy) for learning bounded treewidth BNs from very large datasets. Before performing the structure search, k-greedy initializes a cache of candidate parent sets for each node using the approach of Scanagatta et al. [51]. Then, it samples the space of orderings of variables, performing the next steps for each order. First, an initial structure with the first $k+1$ variables in the order is learned. Depending on the value of k, k-greedy uses either an exact [52] or an approximate [51] structure learning method. Second, the structure incrementally grows according to the chosen order, ensuring that at each step the moral graph of the structure is a partial k-tree. This process is repeated until the maximum allowed execution time is met. Very recently, Scanagatta et al. [53] improved k-greedy by proposing a heuristic score for choosing the order in which the variables are visited. This heuristic ranks the variables by comparing the highest-scoring parent set with the lowest scoring parent set that do not exceed the treewidth bound. The resultant method is called k-MAX. As the former, k-MAX requires predefining a maximum execution time to explore the space of solutions. Extensive experiments showed that both approaches consistently outperform some of the above methods [46,47,49] for learning bounded treewidth BNs. A limitation of k-greedy and k-MAX is that they only learn BNs whose reverse topological order, when used as an EO, has at most width k.

There are also several approaches that learn JTs with bounded treewidth, usually called thin junction trees (TJTs) [54]. This problem is NP-complete when the bound on the treewidth k is greater than one [55]. Chechetka and Guestrin [56] proposed a method that learns TJTs with probably approximately correct (PAC) guarantees in time $O(n^k)$, which is intractable when k is not very small. Shahaf and Guestrin [57] used the graph cuts algorithm [58] to pick the best separator in each iteration during the learning process, requiring polynomial time in both n and k. As mentioned above, heuristics that use

separators usually perform worse in practice than heuristics that search for good EOs for estimating the treewidth of the models.

Some approaches use a penalization in the inference complexity instead of a hard constraint. Lowd and Domingos [59] proposed the first method (LearnAC) to learn arithmetic circuits (ACs)[2] directly from data. This method penalizes the size of each circuit exploiting the local structures of the models to learn networks that can be tractable even for high treewidths. Moves in the space of ACs can be extremely computationally demanding, as circuit structure can be huge. LearnAC uses a greedy approach to address these difficulties, where the best split (i.e., conditioning the conditional probability distribution of a variable to an instance of another variable) is applied at each iteration. Like EOs, the order of splits can have a major effect on network size, and this type of search process is not capable of reconfiguring the split ordering during the learning process. Benjumeda et al. [60] used topological EOs to learn tractable BNs, penalizing each candidate with the width of the EO for the network structure. This method provides for a flexible learning process, accounting for arc additions, removals and reversals. Its main drawback is that the upper bound on the inference complexity provided by topological EOs is not usually as tight as the bound provided by other representations.

3. Elimination trees

This paper addresses the problem of learning bounded treewidth BNs. We focus on choosing a compact representation of the combined space of DAGs and EOs and a set of operators that allow efficiently moving in this space for the next reasons: First, this search space does not put any restrictions on the structure beyond the treewidth bound. Second, given addition, removal and reversal operators, most score+search BN learning methods can be easily adapted to learn bounded treewidth BNs.

In a BN $\mathcal{B}$ over $\mathcal{X} = \{X_1, \ldots, X_n\}$, there are $n!$ different EOs of $\mathcal{X}$, although many are usually equivalent (Definition 3) for the structure $\mathcal{G}$ of $\mathcal{B}$, especially when $\mathcal{G}$ is not densely connected. We need to avoid this redundancy to reduce the size of the search space during the learning process. Next, we define *elimination trees* (ETs), a representation that is especially well suited for this purpose. ETs are based on the representation proposed by Grant and Horsch [14] for recursive conditioning, which we adapt to represent a set $\mathcal{S}$ of EOs for $\mathcal{B}$.

Definition 5. (Elimination tree) Let $\mathcal{B}$ be a BN over $\mathcal{X} = \{X_1, \ldots, X_n\}$. An elimination tree $\mathcal{E}_\mathcal{B}$ over $\mathcal{X}$ is composed of:

- A set of factors or potentials $\phi_{X_1}, \ldots, \phi_{X_n}$ that represent the parameters of $\mathcal{B}$ of each node $X_1, \ldots, X_n$.
- A tree $\mathcal{T}$ composed of a root node, $*$, an inner node (node with parent and children) for each variable $X_i \in \mathcal{X}$, and a leaf node labeled ϕ_{X_i} for each potential ϕ_{X_i}. The nodes are connected by undirected edges.

Assuming that we use VE over an ET $\mathcal{E}_\mathcal{B}$ to perform inference, the topology of the tree shows the orders in which each variable $X_i \in \mathcal{X}$ should be eliminated from the factors of the model. If an inner node X_i is the predecessor (this precedence must be read from the root node to the leaves) of another inner node X_j, X_i is eliminated after X_j.

Definition 6. (ET representation of an EO) Let $\mathcal{B}$ be a BN over $\mathcal{X} = \{X_1, \ldots, X_n\}$. An elimination tree $\mathcal{E}_\mathcal{B}$ represents an EO π for $\mathcal{B}$ if, for each $X_i, X_j \in \mathcal{X}$, $(X_i < X_j)_\pi$ implies that $X_j \notin \mathbf{Desc}_{\mathcal{E}_\mathcal{B}}(X_i)$. $\mathcal{E}_\mathcal{B}$ represents a set of EOs $\mathcal{S}$ for $\mathcal{B}$ if it represents each $\pi_i \in \mathcal{S}$ for $\mathcal{B}$.

Fig. 1 shows an ET $\mathcal{E}_\mathcal{B}$ that represents the set of EOs $\mathcal{S}$ for the probability distribution $P(X_1, X_2, X_3) = \phi_{X_1}(X_1, X_2) \cdot \phi_{X_2}(X_2) \cdot \phi_{X_3}(X_1, X_3)$. As X_1 is a predecessor in $\mathcal{E}_\mathcal{B}$ of X_2 and X_3, $\mathcal{E}_\mathcal{B}$ represents each EO π such that $(X_2 < X_1)_\pi$ and $(X_3 < X_1)_\pi$, that is, (X_2, X_3, X_1) and (X_3, X_2, X_1).

Let us again consider the product of marginals. If we have a BN $\mathcal{B}$ over $\mathcal{X} = \{X_1, \ldots, X_n\}$ that represents the probability distribution $P(X_1, \ldots, X_n) = \phi_{X_1}(X_1) \cdots \phi_{X_n}(X_n)$, all the EOs of $X_1, \ldots, X_n$ are equivalent for $\mathcal{B}$. This can be represented by a single ET, as shown in Fig. 2.

Inference in ETs is straightforward. Given an ET $\mathcal{E}_\mathcal{B}$ that represents a set of EOs $\mathcal{S}$ for $\mathcal{B}$, we could use any EO $\pi_i \in \mathcal{S}$ to perform VE, or to efficiently compile $\mathcal{B}$ into a JT or an AC.

3.1. Properties of elimination trees

In this section, we introduce some terms that we use in the rest of the paper. Let $\mathcal{E}_\mathcal{B}$ be an ET over $\mathcal{X} = \{X_1, \ldots, X_n\}$. We use $\mathrm{Pa}_{\mathcal{E}_\mathcal{B}}(X_i)$ and $\mathbf{Ch}_{\mathcal{E}_\mathcal{B}}(X_i)$ to refer to the parent and the children of node X_i in $\mathcal{E}_\mathcal{B}$. $\mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(X_i)$ and $\mathbf{Desc}_{\mathcal{E}_\mathcal{B}}(X_i)$ refer to the set of predecessor and descendant nodes of X_i in $\mathcal{E}_\mathcal{B}$, respectively. For example, $\mathbf{Desc}_{\mathcal{E}_\mathcal{B}}(X_1) = \{X_2, X_3, \phi_{X_1}, \phi_{X_2}, \phi_{X_3}\}$ and $\mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(X_1) = \{*\}$ in the ET shown in Fig. 3.

[2] ACs are DAGs in which the inner nodes are addition and multiplication nodes and the leaves are numeric variables or constants. They have been adapted to perform inference in BNs [42].

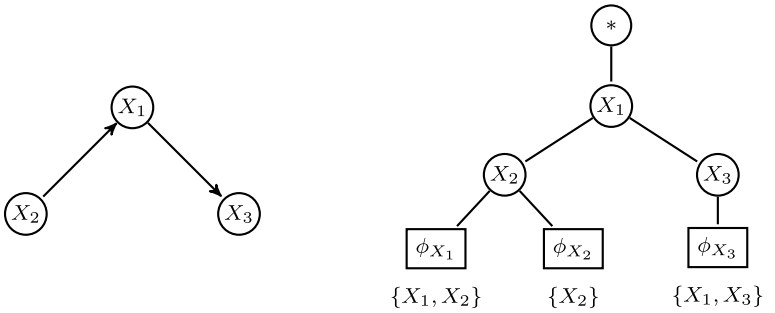


Fig. 1. Structures of a BN $\mathcal{B}$ (left) and an ET $\mathcal{E}_{\mathcal{B}}$ (right). In $\mathcal{E}_{\mathcal{B}}$, $*$ is the root node, X_1, X_2 and X_3 are the inner nodes, and ϕ_{X_1}, ϕ_{X_2} and ϕ_{X_3} are the leaves and potentials. The domain of the potential of each leaf node ϕ_{X_i} is illustrated below the respective node. $\mathcal{E}_{\mathcal{B}}$ represents the EOs (X_2, X_3, X_1) and (X_3, X_2, X_1) for $\mathcal{B}$.

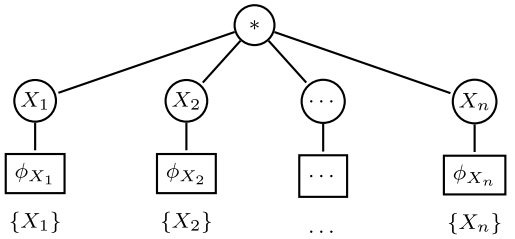


Fig. 2. Structure of an ET that represents the product of marginals. Below each leaf node ϕ_{X_i}, the domain of its corresponding potential is shown.

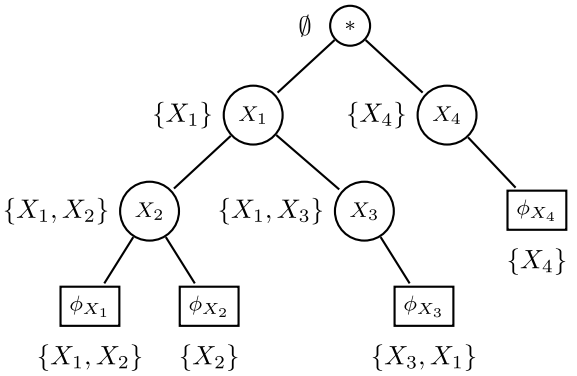


Fig. 3. Structure of an ET. The clusters of the ET are shown near to their corresponding nodes.

Given a factor $\phi_{X_i}(X_{i(1)}, \ldots, X_{i(n_i)})$, $\mathbf{Dom}(\phi_{X_i})$ represents its domain, that is, the set of nodes $\{X_{i(1)}, \ldots, X_{i(n_i)}\}$, where $X_{i(1)}, \ldots, X_{i(n_i)} \in \mathcal{X}$ and n_i is the cardinality of $\mathbf{Dom}(\phi_{X_i})$. We use $\mathbf{Leaves}(\mathcal{E}_{\mathcal{B}})$ to refer to the set of leaf nodes in $\mathcal{E}_{\mathcal{B}}$.

ETs closely resemble dtrees, a representation used for recursive conditioning [38]. Unlike ETs, dtrees are full binary trees (i.e., trees in which any inner node has two children), and their inner nodes are labeled with a set of variables instead of being labeled with a single variable. There follows a definition of clusters in ETs, which is analogous to the definition of clusters given by Darwiche [61] for dtrees.

Definition 7. (Clusters of ET nodes) The cluster of an inner node X_i in an ET $\mathcal{E}_{\mathcal{B}}$ is defined as:

$$\mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_i) := \left(\bigcup_{X_j \in \mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)} \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_j) \setminus \{X_j\} \right).$$

The cluster of a leaf node ϕ_{X_i} in $\mathcal{E}_{\mathcal{B}}$ is defined as:

$$\mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_i}) := \mathbf{Dom}(\phi_{X_i}).$$

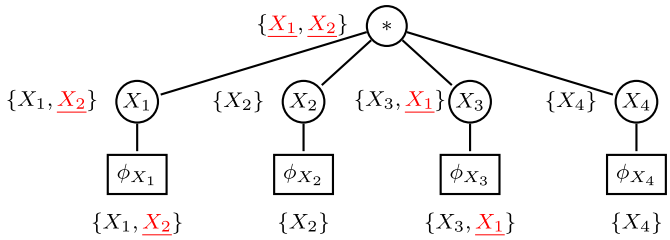


Fig. 4. Structure of an unsound ET. The clusters of the ET are shown near to their respective nodes, and the variables that compromise the soundness of the ET are underlined.

When we perform VE in $\mathcal{E}_\mathcal{B}$, $\mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_i)$ is equivalent to the cluster (domain of the generated factor) induced by eliminating X_i. Fig. 3 shows an example of the clusters $\mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_i)$ of an ET $\mathcal{E}_\mathcal{B}$. The clusters of ETs and the clusters (or cliques) of JTs are also closely related (see Section 3.1.1).

3.1.1. Valid elimination trees

The purpose of using ETs to search for structures with a small treewidth is to reduce the combined space of DAGs and EOs, and consequently allow efficient algorithms for learning bounded treewidth BNs. By the above definition, there are many solutions that are incorrect or redundant. To identify and avoid such ETs during the learning process, we define two new properties: soundness and completeness.

We say that an ET $\mathcal{E}_\mathcal{B}$ is sound if all the EOs that it represents are equivalent for $\mathcal{B}$.

Definition 8. (Sound ETs) Let $\mathcal{E}_\mathcal{B}$ be an ET over $\mathcal{X}$. Node X_i is sound for $\mathcal{E}_\mathcal{B}$ if $\mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_i) \subseteq \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(X_i) \cup \{X_i\}$. A leaf node $\phi_{X_i} \in \mathbf{Leaves}(\mathcal{E}_\mathcal{B})$ is sound for $\mathcal{E}_\mathcal{B}$ if $\mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(\phi_{X_i}) \subseteq \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(\phi_{X_i})$. $\mathcal{E}_\mathcal{B}$ is sound if every node (inner and leaf nodes) is sound for $\mathcal{E}_\mathcal{B}$.

Fig. 4 shows the structure of an unsound ET $\mathcal{E}_\mathcal{B}$. Given that there are no ancestral relationships between the inner nodes in $\mathcal{E}_\mathcal{B}$, it represents all the possible permutations of $\{X_1, X_2, X_3, X_4\}$ as EOs. The clusters of some nodes contain variables (underlined) that are not their predecessors in $\mathcal{E}_\mathcal{B}$. For example, $\mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_1)$ contains X_1 and X_2, but X_2 is not a predecessor of X_1. As there is no ancestral relationship between X_1 and X_2 in $\mathcal{E}_\mathcal{B}$, it is equivalent whether $\mathcal{E}_\mathcal{B}$ eliminates X_1 before or after X_2. Unfortunately, this is not true, as eliminating X_2 before X_1 would induce cluster $\{X_1, X_2\}$. However, this cluster cannot be induced by any EO π_1 where $(X_1 < X_2)_{\pi_1}$ because X_1 will have been eliminated from all factors before X_2 has been eliminated. Thus, if there is a variable that belongs to the cluster of a node X_i that is not one of the predecessors of X_i in $\mathcal{E}_\mathcal{B}$, then the $\mathcal{E}_\mathcal{B}$ is not sound, and it represents EOs that are not equivalent.

The completeness of ETs is analogous to the completeness of a set of EOs.

Definition 9. (Complete ETs) Let $\mathcal{E}_\mathcal{B}$ be an ET over $\mathcal{X}$. Node $X_i \in \mathcal{X} \cup \mathbf{Leaves}(\mathcal{E}_\mathcal{B})$ (i.e., X_i is either an inner node from $\mathcal{X}$ or a leaf node from $\mathbf{Leaves}(\mathcal{E}_\mathcal{B})$) is complete for $\mathcal{E}_\mathcal{B}$ if $\mathrm{Pa}_{\mathcal{E}_\mathcal{B}}(X_i) \in \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_i)$ or $\mathrm{Pa}_{\mathcal{E}_\mathcal{B}}(X_i) = *$. $\mathcal{E}_\mathcal{B}$ is complete if every node (inner and leaf nodes) is complete for $\mathcal{E}_\mathcal{B}$.

Fig. 5 shows the structure of an incomplete ET $\mathcal{E}_\mathcal{B}$. It represents the EOs $\mathcal{S} = \{(X_3, X_2, X_1, X_4), (X_3, X_2, X_4, X_1), (X_3, X_4, X_2, X_1), (X_4, X_3, X_2, X_1)\}$, but there are other EOs that are equivalent for $\mathcal{B}$ that are not represented by $\mathcal{E}_\mathcal{B}$. For example, (X_2, X_3, X_1, X_4) is equivalent to (X_3, X_2, X_1, X_4) given that the clusters induced after eliminating X_2 and X_3 are $\{X_1, X_2\}$ and $\{X_1, X_3\}$ in both cases.

Definition 10. (Valid ETs) An ET $\mathcal{E}_\mathcal{B}$ is valid if it is sound and complete.

The ET shown in Fig. 3 is sound (for every node X_i, all the variables in its cluster are either its predecessors or X_i) and complete (for every node X_i with parent X_p, the cluster of X_i contains X_p). This means that it is valid. The space of valid ETs does not contain incorrect or redundant solutions.

The process described by Algorithm 1 yields a valid ET $\mathcal{E}_\mathcal{B}$ given a BN $\mathcal{B}$ and an EO π.

Algorithm 1 starts with an ET where the parent of every node is the root node $*$ (line 1). First, the variables in $\mathcal{X}$ are visited in the order given by π (line 2). When variable X_i is visited, node X_i is set as the parent of the nodes whose cluster contains X_i and whose parent is the root node $*$ in the ET (lines 3–7). This is analogous to the process of eliminating variable X_i from $\mathcal{B}$. The cluster $\mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_i)$ of X_i in the ET $\mathcal{E}_\mathcal{B}$ is output in the same way as the cluster $\mathbf{Cls}_\pi(X_i)$ induced by eliminating X_i using π in $\mathcal{B}$, and they are equal.

Proposition 1 states that given an EO π and a BN $\mathcal{B}$, Algorithm 1 returns always valid ETs. Hence, there is at least one valid ET for π and $\mathcal{B}$.

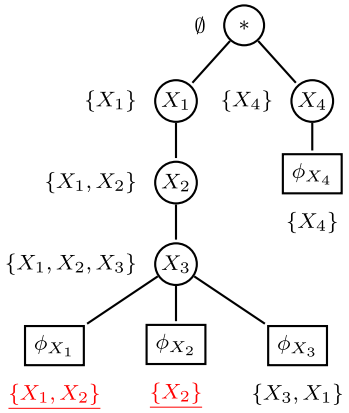


Fig. 5. Structure of an incomplete ET. The clusters of the ET are shown near to their respective nodes, and the clusters that compromise the completeness of the ET are underlined.

Input: BN $\mathcal{B}$ over $\mathcal{X}$, EO π
Output: Valid ET $\mathcal{E}_{\mathcal{B}}$
1 let $\mathcal{E}_{\mathcal{B}}$ be an ET with inner nodes $\mathcal{X}$ and factor nodes **Leaves**$(\mathcal{E}_{\mathcal{B}})$ where $\forall X_i \in \mathcal{X} \cup$ **Leaves**$(\mathcal{E}_{\mathcal{B}})$, $\text{Pa}_{\mathcal{E}_{\mathcal{B}}}(X_i) = *$;
2 **for** $X_i \in \pi$ **do**
3 **for** $X_j \in (\mathcal{X} \cup \textit{Leaves}(\mathcal{E}_{\mathcal{B}})) \cap \textit{Ch}_{\mathcal{E}_{\mathcal{B}}}(*)$ **do**
4 **if** $X_i \in \textit{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_j)$ **then**
5 $\text{Pa}_{\mathcal{E}_{\mathcal{B}}}(X_j) \leftarrow X_i$;
6 **end**
7 **end**
8 **end**
9 **return** $\mathcal{E}_{\mathcal{B}}$;

Algorithm 1: Compile a valid ET from an EO and a BN.

Proposition 1. *Let $\mathcal{B}$ be a BN over $\mathcal{X}$ and π an EO of $\mathcal{X}$. Algorithm 1 returns a valid ET that represents π for $\mathcal{B}$.*

Proof. Algorithm 1 ensures that when any variable X_i is visited, it is set as the parent of every node whose cluster contains X_i and whose parent is the root node $*$. Therefore:

- If the cluster of node X_j contains X_i when X_i is visited, the cluster of each of its predecessors also has X_i, given that node X_i has no children until X_i has been visited. When X_i is visited, it is set as a predecessor of all the nodes whose cluster contains X_i. After visiting X_i, there are no nodes whose clusters contain X_i that are not their descendants in $\mathcal{E}_{\mathcal{B}}$. As this applies to each node $X_i \in \mathcal{X}$, all the nodes in $\mathcal{E}_{\mathcal{B}}$ must be sound, making $\mathcal{E}_{\mathcal{B}}$ sound.
- The cluster of every node X_j that is a child of X_i contains X_i. Each X_j is complete for $\mathcal{E}_{\mathcal{B}}$, making $\mathcal{E}_{\mathcal{B}}$ complete.
- A node X_j can only be a descendant of a node X_i in $\mathcal{E}_{\mathcal{B}}$ if $(X_j < X_i)_{\pi}$. Hence, $\mathcal{E}_{\mathcal{B}}$ represents π.

As $\mathcal{E}_{\mathcal{B}}$ is valid (sound and complete) and represents π, there is at least one valid $\mathcal{E}_{\mathcal{B}}$ for $\mathcal{B}$ and π. □

Proposition 2 ensures that there is a single valid ET $\mathcal{E}_{\mathcal{B}}$ that represents an EO π for a BN $\mathcal{B}$.

Proposition 2. *Let $\mathcal{B}$ be a BN over $\mathcal{X}$ and π an EO of $\mathcal{X}$. There is exactly one valid ET $\mathcal{E}_{\mathcal{B}}$ that represents π for $\mathcal{B}$.*

Proof. From Proposition 1, we know that there is at least one valid ET for $\mathcal{B}$ and π. We prove that there is exactly one by structural induction. We consider two ETs $\mathcal{E}^1_{\mathcal{B}}$ and $\mathcal{E}^2_{\mathcal{B}}$ for $\mathcal{B}$ and π. We show that if $\mathcal{E}^1_{\mathcal{B}}$ and $\mathcal{E}^2_{\mathcal{B}}$ are valid, then, for each node $X_i \in \mathcal{X}$, $\text{Pa}_{\mathcal{E}^2_{\mathcal{B}}}(X_i) = \text{Pa}_{\mathcal{E}^1_{\mathcal{B}}}(X_i)$, starting from the leaves (base case). This means that $\mathcal{E}^1_{\mathcal{B}}$ and $\mathcal{E}^2_{\mathcal{B}}$ are the same, which implies that there is a single valid ET for $\mathcal{B}$ and π.

Base case:

The subtrees that have $\phi_{X_i} \in$ **Leaves**$(\mathcal{E}_{\mathcal{B}})$ as its root in $\mathcal{E}^1_{\mathcal{B}}$ and $\mathcal{E}^2_{\mathcal{B}}$ are only composed of node ϕ_{X_i}. Hence, they are equal.

Inductive step:

Assume that the subtrees hanging from node X_i in $\mathcal{E}^1_{\mathcal{B}}$ and $\mathcal{E}^2_{\mathcal{B}}$ are equal. Let $X_j = \text{Pa}_{\mathcal{E}^1_{\mathcal{B}}}(X_i)$ (Fig. 6). As $\mathcal{E}^1_{\mathcal{B}}$ is valid, if $X_j \neq \text{Pa}_{\mathcal{E}^2_{\mathcal{B}}}(X_i)$, then $\text{Pa}_{\mathcal{E}^2_{\mathcal{B}}}(X_i)$ is a node X_k where either:

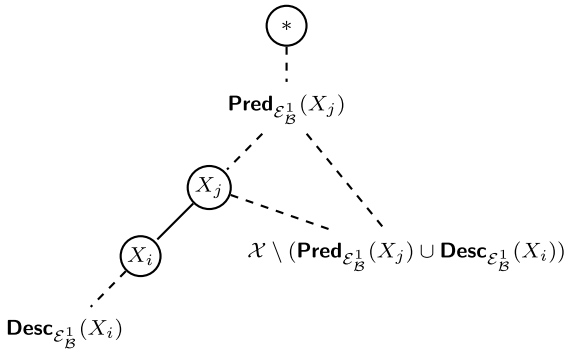


Fig. 6. State of the ET $\mathcal{E}^1_{\mathcal{B}}$ during the inductive step of the proof of Propositions 2 and 3.

a) $X_k \in \mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}}}(X_j)$: then $\mathcal{E}^2_{\mathcal{B}}$ would not represent π because $(X_j < X_k)_\pi$, or
b) $X_k \in \mathbf{Desc}_{\mathcal{E}^1_{\mathcal{B}}}(X_i)$: then $\mathcal{E}^2_{\mathcal{B}}$ would not represent π because $(X_k < X_i)_\pi$, or
c) $X_k \in \mathcal{X} \setminus (\mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}}}(X_j) \cup \mathbf{Desc}_{\mathcal{E}^1_{\mathcal{B}}}(X_i))$: then $\mathcal{E}^2_{\mathcal{B}}$ is not complete because $\mathrm{Pa}_{\mathcal{E}^2_{\mathcal{B}}}(X_i) = X_j \notin \mathbf{Cls}_{\mathcal{E}^2_{\mathcal{B}}}(X_i)$.

This means that if $\mathcal{E}^2_{\mathcal{B}}$ is valid, then the following condition $\mathrm{Pa}_{\mathcal{E}^1_{\mathcal{B}}}(X_i) = \mathrm{Pa}_{\mathcal{E}^2_{\mathcal{B}}}(X_i)$ holds. □

Valid ETs avoid the redundancy between EOs and DAGs. We demonstrate that a valid ET $\mathcal{E}_{\mathcal{B}}$ represents a complete set of equivalent EOs for $\mathcal{B}$.

Proposition 3. *Let $\mathcal{B}$ be a BN over $\mathcal{X}$, π an EO of $\mathcal{X}$, and $\mathcal{E}^1_{\mathcal{B}}$ a valid ET that represents π_1 for $\mathcal{B}$. For each EO π_2 equivalent to π_1 for $\mathcal{B}$ (Definition 3), $\mathcal{E}^1_{\mathcal{B}}$ also represents π_2.*

Proof. Let $\mathbf{Cl}_i^{\pi_1}$ and $\mathbf{Cl}_i^{\pi_2}$ be the cluster induced by VE after eliminating X_i from $\mathcal{B}$ using the EO π_1 and π_2, respectively. As π_1 and π_2 are equivalent for $\mathcal{B}$, $\mathbf{Cl}_i^{\pi_1} = \mathbf{Cl}_i^{\pi_2}$.

From Proposition 2 we know that there is a single valid ET $\mathcal{E}^1_{\mathcal{B}}$ that represents π_1 for $\mathcal{B}$, and also a single valid ET $\mathcal{E}^2_{\mathcal{B}}$ that represents π_2 for $\mathcal{B}$. We prove that $\mathcal{E}^1_{\mathcal{B}}$ also represents π_2 by structural induction. We show that if $\mathcal{E}^1_{\mathcal{B}}$ and $\mathcal{E}^2_{\mathcal{B}}$ are valid, then, for each of node $X_i \in \mathcal{X}$, starting from the leaves (base case), $\mathrm{Pa}_{\mathcal{E}^1_{\mathcal{B}}}(X_i) = \mathrm{Pa}_{\mathcal{E}^2_{\mathcal{B}}}(X_i)$. This means that $\mathcal{E}^1_{\mathcal{B}}$ and $\mathcal{E}^2_{\mathcal{B}}$ are the same, which implies that $\mathcal{E}^1_{\mathcal{B}}$ also represents π_2 for $\mathcal{B}$.

Base case:

The subtrees whose root is $\phi_{X_i} \in \mathbf{Leaves}(\mathcal{E}^1_{\mathcal{B}})$ in $\mathcal{E}^1_{\mathcal{B}}$ and $\mathcal{E}^2_{\mathcal{B}}$ are composed of node ϕ_{X_i} only. Hence, they are equal.

Inductive step:

Assume that the subtrees hanging from node X_i in $\mathcal{E}^1_{\mathcal{B}}$ and $\mathcal{E}^2_{\mathcal{B}}$ are equal. Let $X_j = \mathrm{Pa}_{\mathcal{E}^1_{\mathcal{B}}}(X_i)$ (Fig. 6). As $\mathcal{E}^1_{\mathcal{B}}$ is valid, if $X_j \neq \mathrm{Pa}_{\mathcal{E}^2_{\mathcal{B}}}(X_i)$, then $\mathrm{Pa}_{\mathcal{E}^2_{\mathcal{B}}}(X_i)$ is a node X_k where either:

a) $X_k \in \mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}}}(X_j)$: then $X_j \notin \mathbf{Cls}_{\pi_1}(X_k)$ given that $X_j \in \mathbf{Desc}_{\mathcal{E}^1_{\mathcal{B}}}(X_k)$. Assuming that $\mathbf{Cls}_{\pi_1}(X_i) = \mathbf{Cls}_{\pi_2}(X_i)$, $X_j \in \mathbf{Cls}_{\pi_2}(X_k)$ given that $X_j \in \mathbf{Cls}_{\pi_2}(X_i) = \mathbf{Cls}_{\pi_1}(X_i)$ (X_j is the parent of X_i in $\mathcal{E}^2_{\mathcal{B}}$) and that X_k is the parent of X_i in $\mathcal{E}^2_{\mathcal{B}}$. Hence, $\mathbf{Cls}_{\pi_1}(X_k) \neq \mathbf{Cls}_{\pi_2}(X_k)$.
b) $X_k \notin \mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}}}(X_j)$: then $X_k \notin \mathbf{Cls}_{\pi_1}(X_i)$ given that $X_i \notin \mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}}}(X_k)$, and $X_k \in \mathbf{Cls}_{\pi_2}(X_i)$ given that $\mathrm{Pa}_{\mathcal{E}^2_{\mathcal{B}}}(X_i) = X_k$. Thus, $\mathbf{Cls}_{\pi_1}(X_i) \neq \mathbf{Cls}_{\pi_2}(X_i)$.

This means that if $\mathcal{E}^1_{\mathcal{B}}$ is valid, then the following condition $\mathrm{Pa}_{\mathcal{E}^1_{\mathcal{B}}}(X_i) = \mathrm{Pa}_{\mathcal{E}^2_{\mathcal{B}}}(X_i)$ holds. □

Additionally, valid ETs can be easily transformed into JTs. Let $C_1, \ldots, C_n$ be the cluster sequence induced by VE in a BN $\mathcal{B}$ with an EO π. The maximal clusters (clusters that are not contained in other clusters) in $C_1, \ldots, C_n$ can be connected to form a JT with the same width as π [61]. $C_1, \ldots, C_n$ are also the clusters of the inner nodes of the valid ET$\mathcal{E}_{\mathcal{B}}$ that represents π for $\mathcal{B}$. This means that the maximal clusters of an ET can be connected to create a JT of the same width.

3.2. Inference complexity in elimination trees

Given the above definitions it is simple to analyze inference complexity in ETs, which we will later use to define the proposed algorithm for learning bounded treewidth BNs. Inference by VE is exponential in the width of the chosen EO. Analogously, inference in an ET $\mathcal{E}_{\mathcal{B}}$ is exponential in the width of $\mathcal{E}_{\mathcal{B}}$.

Definition 11. (ET width) The width of an ET $\mathcal{E}_{\mathcal{B}}$ is the length of its largest cluster minus one.

Given a BN $\mathcal{B}$, the width of the ET $\mathcal{E}_{\mathcal{B}}$ with lowest width that is valid for $\mathcal{B}$ is the treewidth of $\mathcal{B}$. Therefore, if $\mathcal{E}_{\mathcal{B}}$ is good enough (near-minimum width), its width is an indicator of the inference complexity of the model as it should be close to the treewidth of $\mathcal{B}$.

4. Learning elimination trees

4.1. Compiling changes

A naive solution to learn bounded treewidth BNs would be to use a known heuristic to output a good EO (see Section 2.2). The width of the chosen EO is an estimate of the treewidth of the BN candidates. However, it is computationally demanding to search for good EOs from scratch, and it can be intractable if we have to perform this process for each candidate during the structure search. The results shown in Section 3 can be applied to learn tractable BNs in the combined space of DAGs and EOs. Our proposal is to limit the treewidth of the BN by bounding the width of the ET (Definition 11). This strategy requires obtaining a valid ET for each network candidate during the learning process.

In this section, we propose methods to compile incrementally in ETs the arc additions and removals made to a BN during the learning process, and show that the proposed algorithms always output valid ETs. As the reversal of arc $X_{\text{out}} \to X_{\text{in}}$ in $\mathcal{B}$ can be seen as the removal of arc $X_{\text{out}} \to X_{\text{in}}$ followed by the addition of the reversed arc $X_{\text{in}} \to X_{\text{out}}$, we assume that both changes are compiled each time a reversal is made to a BN.

4.1.1. Arc addition

The addition of an arc $X_{\text{out}} \to X_{\text{in}}$ in $\mathcal{B}$ may compromise the soundness of an ET $\mathcal{E}_{\mathcal{B}}$. If $\mathcal{E}_{\mathcal{B}}$ is valid, it represents a complete set of equivalent EOs $\mathcal{S}$ (Proposition 2). The addition of $X_{\text{out}} \to X_{\text{in}}$ in $\mathcal{B}$ places a new restriction on the equivalence of the EOs in $\mathcal{S}$. After applying this local change, there is at least one factor over both X_{out} and X_{in}. Therefore, an ET $\mathcal{E}'_{\mathcal{B}'}$ can only be valid for the new BN $\mathcal{B}'$ if it encodes an ancestral relationship between X_{out} and X_{in}. Algorithm 2 modifies the structure of $\mathcal{E}_{\mathcal{B}}$ to meet the new restrictions. The resulting ET $\mathcal{E}'_{\mathcal{B}'}$ represents a complete subset of EOs $\mathcal{S}' \subseteq \mathcal{S}$ (see Fig. 7) that are also equivalent in $\mathcal{B}'$.

Algorithm 2 receives a valid ET $\mathcal{E}_{\mathcal{B}}$, and arc addition $X_{\text{out}} \to X_{\text{in}}$. In $\mathcal{B}'$, variable X_{out} is added to the domain of $\phi_{X_{\text{in}}}$ (line 2). The clusters of the nodes that are predecessors of $\phi_{X_{\text{in}}}$ and descendants of X_{out} contain X_{out} after applying this change. There are three different scenarios that require performing different changes in the ET to ensure its validity.

If X_{out} is a predecessor of $\phi_{X_{\text{in}}}$ in $\mathcal{E}'_{\mathcal{B}'}$, it is not necessary to make any changes in $\mathcal{E}'_{\mathcal{B}'}$. Otherwise, some nodes contain X_{out} in their clusters but not in their predecessors, and $\mathcal{E}'_{\mathcal{B}'}$ is not sound. If $X_f = \text{Pa}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}})$ is a predecessor of X_{out} in $\mathcal{E}_{\mathcal{B}}$ (line 4), X_{out} is set as the new parent of $\phi_{X_{\text{in}}}$ in $\mathcal{E}'_{\mathcal{B}'}$ (line 5). Thus, X_{out} is a predecessor of $\phi_{X_{\text{in}}}$ in $\mathcal{E}'_{\mathcal{B}'}$.

Input: Valid ET $\mathcal{E}_{\mathcal{B}}$, output node X_{out}, input node X_{in}
Output: Valid ET $\mathcal{E}'_{\mathcal{B}'}$

1 let $\mathcal{E}'_{\mathcal{B}'}$ be a copy of $\mathcal{E}_{\mathcal{B}}$;
2 $\mathbf{Dom}(\phi_{X_{\text{in}}}) \leftarrow \mathbf{Dom}(\phi_{X_{\text{in}}}) \cup \{X_{\text{out}}\}$;
3 $X_f \leftarrow \text{Pa}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}})$;
4 **if** $X_f \in \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}})$ **then**
5 $\quad \text{Pa}_{\mathcal{E}'_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) \leftarrow X_{\text{out}}$;
6 **else if** $X_{\text{out}} \notin \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}})$ **then**
7 $\quad$ let X_m be the deepest node in $\mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}'}}(X_{\text{out}}) \cap \mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}'}}(\phi_{X_{\text{in}}})$;
8 $\quad$ let $\mathcal{E}^1_{\mathcal{B}'}$ and $\mathcal{E}^2_{\mathcal{B}'}$ be two copies of $\mathcal{E}'_{\mathcal{B}'}$;
9 $\quad X_k \leftarrow \mathbf{Ch}_{\mathcal{E}^1_{\mathcal{B}'}}(X_m) \cap \mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}'}}(\phi_{X_{\text{in}}})$;
10 $\quad \text{Pa}_{\mathcal{E}^1_{\mathcal{B}'}}(X_k) \leftarrow X_{\text{out}}$;
11 $\quad X_h \leftarrow \mathbf{Ch}_{\mathcal{E}^2_{\mathcal{B}'}}(X_m) \cap \mathbf{Pred}_{\mathcal{E}^2_{\mathcal{B}'}}(X_{\text{out}})$;
12 $\quad \text{Pa}_{\mathcal{E}^2_{\mathcal{B}}}(X_h) \leftarrow X_f$;
13 $\quad \text{Pa}_{\mathcal{E}^2_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \leftarrow X_{\text{out}}$;
14 $\quad$ **if** $\text{width}(\mathcal{E}^1_{\mathcal{B}'}) < \text{width}(\mathcal{E}^2_{\mathcal{B}'})$ **then**
15 $\quad\quad \mathcal{E}'_{\mathcal{B}'} \leftarrow \mathcal{E}^1_{\mathcal{B}'}$;
16 $\quad$ **else**
17 $\quad\quad \mathcal{E}'_{\mathcal{B}'} \leftarrow \mathcal{E}^2_{\mathcal{B}'}$;
18 $\quad$ **end**
19 **end**
20 **return** $\mathcal{E}'_{\mathcal{B}'}$;

Algorithm 2: Compilation of the addition of arc $X_{\text{out}} \to X_{\text{in}}$ ($\text{add}(\mathcal{E}_{\mathcal{B}}, X_{\text{out}}, X_{\text{in}})$).

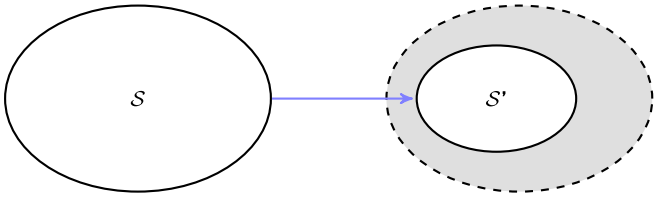


Fig. 7. A set of EOs $\mathcal{S}$ that are equivalent for a BN $\mathcal{B}$, and a subset $\mathcal{S}'$ of $\mathcal{S}$ that are equivalent for the BN $\mathcal{B}'$, output after adding an arc in $\mathcal{B}$.

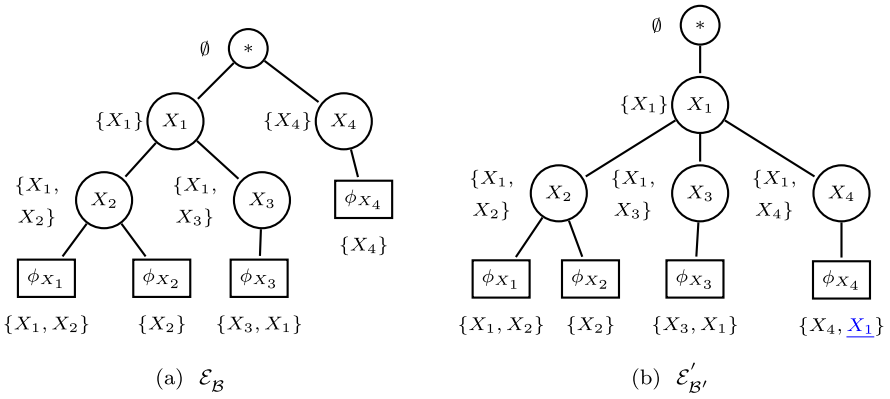


Fig. 8. (a) an ET $\mathcal{E}_{\mathcal{B}}$, and (b) the ET $\mathcal{E}'_{\mathcal{B}'}$ output after incrementally compiling the arc addition $X_1 \to X_4$ (i.e., addition of X_1 to $\mathbf{Dom}(\phi_{X_4})$) in $\mathcal{E}_{\mathcal{B}}$.

If X_{out} is not a predecessor of $\phi_{X_{\text{in}}}$ in $\mathcal{E}_{\mathcal{B}}$ and X_f is not a predecessor of X_{out} in $\mathcal{E}_{\mathcal{B}}$ (line 6), the cluster of the node in $\{X_{\text{out}}\} \cup \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \cap \mathbf{Desc}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}})$ contains X_{out} but the clusters of their predecessors in $\mathcal{E}_{\mathcal{B}}$ do not. Algorithm 2 creates two candidate ETs ($\mathcal{E}^1_{\mathcal{B}'}$ and $\mathcal{E}^2_{\mathcal{B}'}$). The first is output by setting X_{out} as the parent of X_k in $\mathcal{E}^1_{\mathcal{B}'}$ (line 10), that is, the shallowest predecessor of $\phi_{X_{\text{in}}}$ in $\mathcal{E}_{\mathcal{B}}$ that does not belong to $\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}})$ (line 9). The second is output by setting $\text{Pa}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}})$ as the parent of X_h in $\mathcal{E}^2_{\mathcal{B}'}$ (line 12), that is, the shallowest predecessor of X_{out} in $\mathcal{E}_{\mathcal{B}}$ that does not belong to $\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}})$ (line 11), and setting X_{out} as the new parent of $\phi_{X_{\text{in}}}$ in $\mathcal{E}^2_{\mathcal{B}'}$ (line 13). $\mathcal{E}'_{\mathcal{B}'}$ is selected as the ET of smaller width between $\mathcal{E}^1_{\mathcal{B}'}$ and $\mathcal{E}^2_{\mathcal{B}'}$ (lines 14–18). Either way, X_{out} is a predecessor in $\mathcal{E}'_{\mathcal{B}'}$ of the nodes in $\{X_{\text{out}}\} \cup \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \cap \mathbf{Desc}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}})$, and the returned ET $\mathcal{E}'_{\mathcal{B}'}$ is valid (Lemma 1).

Lemma 1. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET that represents $\mathcal{B}$ over $\mathcal{X}$. Given $X_{out}, X_{in} \in \mathcal{X}$, the ET $\mathcal{E}'_{\mathcal{B}'}$ yielded after applying add($\mathcal{E}_{\mathcal{B}}, X_{out}, X_{in}$) in Algorithm 2, is also a valid ET representing $\mathcal{B}'$ over $\mathcal{X}$.*

Proof. See Appendix A. □

This process can be performed efficiently in ETs of bounded width (see Theorem 2).

Fig. 8 shows an example of the incremental compilation of the addition of arc $X_1 \to X_4$ in $\mathcal{X}_2$. $\mathcal{E}_{\mathcal{B}}$ represents all the permutations of $\{X_1, X_2, X_3, X_4\}$ where X_2 and X_3 are eliminated before X_1. After adding $X_1 \to X_4$ in $\mathcal{B}$, there are EOs represented by $\mathcal{E}_{\mathcal{B}}$ that are not equivalent for $\mathcal{B}'$ (e.g., $\{X_2, X_3, X_1, X_4\}$ and $\{X_2, X_3, X_4, X_1\}$). Using Algorithm 2, $X_{\text{out}} = X_1$ and $X_{\text{in}} = X_4$. As $X_f = X_4$ is not a predecessor of X_1 in $\mathcal{E}_{\mathcal{B}}$ (line 4), and X_1 is not a predecessor of ϕ_{X_4}(line 6), two ETs $\mathcal{E}^1_{\mathcal{B}'}$ and $\mathcal{E}^2_{\mathcal{B}'}$ are created. Assuming that $\mathcal{E}^1_{\mathcal{B}'}$ is smaller than $\mathcal{E}^2_{\mathcal{B}'}$, X_1 is the new parent of X_4 in $\mathcal{E}'_{\mathcal{B}'}$.

4.1.2. Arc removal

The removal of an arc $X_{\text{out}} \to X_{\text{in}}$ in $\mathcal{B}$ may compromise the completeness of an ET $\mathcal{E}_{\mathcal{B}}$. Let $\mathcal{S}$ be the set of EOs represented by $\mathcal{E}_{\mathcal{B}}$. The removal of $X_{\text{out}} \to X_{\text{in}}$ in $\mathcal{B}$ leads to a reduction in the restrictions on EO equivalence in $\mathcal{S}$. This means that $\mathcal{E}_{\mathcal{B}}$ may not represent all the EOs that are equivalent to EOs in $\mathcal{S}$. Algorithm 3 yields an ET that represents a complete superset of EOs $\mathcal{S}' \supseteq \mathcal{S}$ (see Fig. 9) containing all the EOs that are equivalent to EOs in $\mathcal{S}$ for $\mathcal{B}'$, which is the resulting BN after removing arc $X_{\text{out}} \to X_{\text{in}}$ from $\mathcal{B}$.

After removing $X_{\text{out}} \to X_{\text{in}}$ from $\mathcal{B}$, the shallowest node in $(\mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) \cup \{\phi_{X_{\text{in}}}\}) \cap \mathbf{Desc}_{\mathcal{E}'_{\mathcal{B}'}}(X_{\text{out}})$, which we refer to as X_i (line 4), may not be complete. If X_i is not complete (line 5), Algorithm 3 sets the deepest node in $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}'}}(X_i)$, which we refer to as X'_j, as its new parent (lines 6–7). Note that the idea behind this change is that the new parent of X_i is in its cluster, making X_i complete. After this change, the shallowest node in $\{X_j\} \cup \mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}'}}(X_j) \cap \mathbf{Desc}_{\mathcal{E}'_{\mathcal{B}'}}(X'_j)$ may not be sound.

Input: Valid ET $\mathcal{E}_{\mathcal{B}}$, output node X_{out}, input node X_{in}
Output: Valid ET $\mathcal{E}'_{\mathcal{B}'}$
1 let $\mathcal{E}'_{\mathcal{B}'}$ be a copy of $\mathcal{E}_{\mathcal{B}}$;
2 $\mathbf{Dom}(\phi(X_{\text{in}})) \leftarrow \mathbf{Dom}(\phi(X_{\text{in}})) \setminus \{X_{\text{out}}\}$;
3 $X_j \leftarrow X_{\text{out}}$;
4 let X_i be the shallowest node in $(\mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) \cup \{\phi_{X_{\text{in}}}\}) \cap \mathbf{Desc}_{\mathcal{E}'_{\mathcal{B}'}}(X_{\text{out}})$;
5 **while** $X_j \notin \mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}'}}(X_i)$ **and** $X_j \neq *$ **do**
6 let X'_j be the deepest node in $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}'}}(X_i) \setminus \{X_i\}$ if $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}'}}(X_i) \setminus \{X_i\} \neq \emptyset$ and $*$ otherwise ;
7 $\text{Pa}_{\mathcal{E}'_{\mathcal{B}'}}(X_i) \leftarrow X'_j$;
8 set X_i as the shallowest node in $\{X_j\} \cup \mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}'}}(X_j) \cap \mathbf{Desc}_{\mathcal{E}'_{\mathcal{B}'}}(X'_j)$;
9 $X_j \leftarrow X'_j$;
10 **end**
11 **return** $\mathcal{E}'_{\mathcal{B}'}$;

Algorithm 3: Compilation of the removal of arc $X_{\text{out}} \rightarrow X_{\text{in}}$ (remove($\mathcal{E}_{\mathcal{B}}, X_{\text{out}}, X_{\text{in}}$)).

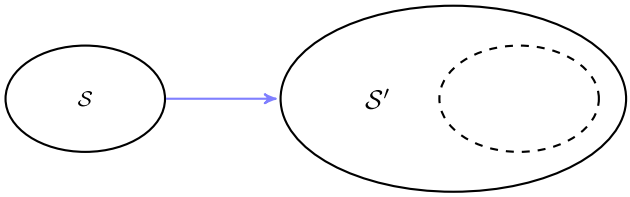


Fig. 9. A set of EOs $\mathcal{S}$ that are equivalent for a BN $\mathcal{B}$, and a superset $\mathcal{S}'$ of $\mathcal{S}$ that are equivalent for the BN $\mathcal{B}'$, yielded after removing an arc in $\mathcal{B}$.

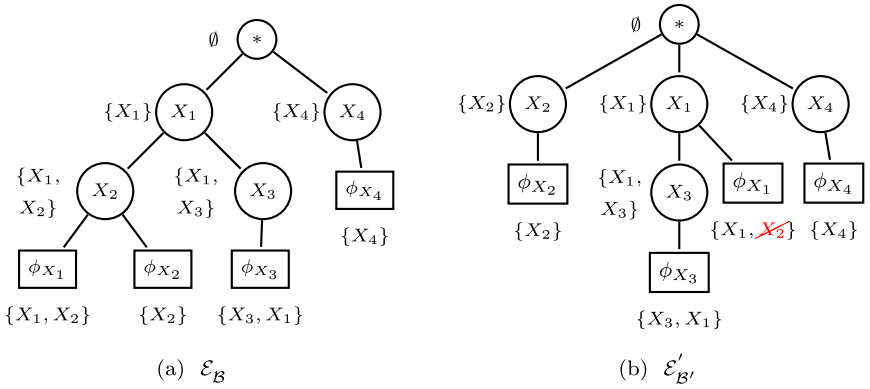


Fig. 10. (a) an ET $\mathcal{E}_{\mathcal{B}}$, and (b) an ET $\mathcal{E}'_{\mathcal{B}'}$ yielded after incrementally compiling the removal of arc $X_2 \rightarrow X_1$ (i.e., removal of X_2 from $\mathbf{Dom}(\phi_{X_1})$) in $\mathcal{E}_{\mathcal{B}}$.

Thus, Algorithm 3 repeats the same process until every node is complete, guaranteeing the validity of every node in $\mathcal{E}'_{\mathcal{B}'}$ (Lemma 2).

Lemma 2. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET that represents $\mathcal{B}$ over $\mathcal{X}$. Given $X_{out}, X_{in} \in \mathcal{X}$, the ET $\mathcal{E}'_{\mathcal{B}'}$ that represents $\mathcal{B}'$ output after applying remove($\mathcal{E}_{\mathcal{B}}, X_{out}, X_{in}$) in Algorithm 3, is also valid.*

Proof. See Appendix A. □

Fig. 10 shows an example of how Algorithm 3 compiles, in an ET $\mathcal{E}_{\mathcal{B}}$, the removal of arc $X_2 \rightarrow X_1$ in $\mathcal{B}$. Let $\mathcal{E}'_{\mathcal{B}'}$ be a copy of $\mathcal{E}_{\mathcal{B}}$ where X_2 has been removed from the domain of ϕ_{X_1}. First, X_j is set to X_{out} (line 3). As $X_{\text{out}} = X_2$ and $X_{\text{in}} = X_1$, X_i is the shallowest node in $\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_1}) \cup \{X_1\}$ that is a descendant of X_2 (line 4), namely ϕ_{X_1}.

X_{out} is not in the cluster of ϕ_{X_1} in $\mathcal{E}'_{\mathcal{B}'}$ (line 5). Hence, X'_j is set to X_1, that is, the deepest node in the cluster of ϕ_{X_1} (line 6). X_1 is now set as the parent of ϕ_{X_1} in $\mathcal{E}'_{\mathcal{B}'}$, which makes node ϕ_{X_1} complete. The new X_i is set to X_2 (line 8), and the new X_j is X_1 (line 9).

Input: Valid ET $\mathcal{E}_{\mathcal{B}}$, node X_i
Output: Valid ET $\mathcal{E}'_{\mathcal{B}}$
1 Let $\mathcal{E}'_{\mathcal{B}}$ be a copy of $\mathcal{E}_{\mathcal{B}}$;
2 $X_j \leftarrow \mathrm{Pa}_{\mathcal{E}'_{\mathcal{B}}}(X_i)$;
3 $\mathrm{Pa}_{\mathcal{E}'_{\mathcal{B}}}(X_i) \leftarrow \mathrm{Pa}_{\mathcal{E}'_{\mathcal{B}}}(X_j)$;
4 $\mathrm{Pa}_{\mathcal{E}'_{\mathcal{B}}}(X_j) \leftarrow X_i$;
5 **for** $X_k \in \mathbf{Ch}_{\mathcal{E}'_{\mathcal{B}}}(X_i)$ **do**
6 | **if** $X_j \in \mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}}}(X_k)$ **then**
7 | | $\mathrm{Pa}_{\mathcal{E}'_{\mathcal{B}}}(X_k) \leftarrow X_j$;
8 | **end**
9 **end**
10 **return** $\mathcal{E}'_{\mathcal{B}}$;

Algorithm 4: Swap of X_i and $\mathrm{Pa}_{\mathcal{E}_{\mathcal{B}}}(X_i)$ in ET $\mathcal{E}_{\mathcal{B}}$ (swap($\mathcal{E}_{\mathcal{B}}, X_i$)).

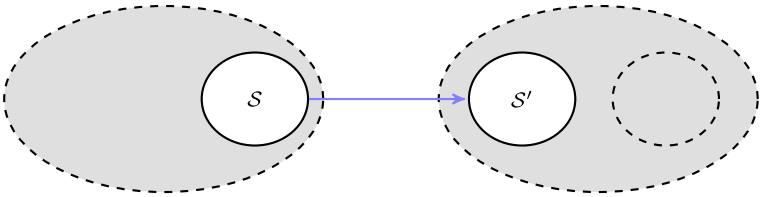


Fig. 11. The EOs in $\mathcal{S}$ are equivalent for a BN $\mathcal{B}$, and a swap produces other set of EOs $\mathcal{S}'$, with $\mathcal{S} \cap \mathcal{S}' = \emptyset$, where the EOs in $\mathcal{S}'$ are also equivalent for $\mathcal{B}$.

As X_1 is not in the cluster of X_2 in $\mathcal{E}_{\mathcal{B}}$, the same process is applied again (lines 6–9). In this case, the root node $*$ is set as the parent of X_2 in $\mathcal{E}'_{\mathcal{B}'}$ and the new X_j is $*$, ending the loop (line 5) and returning $\mathcal{E}'_{\mathcal{B}'}$.

The complexity of this process is polynomial in the number of variables and the width of the ET (see Theorem 2).

4.2. Optimization

The methods described above adapt an ET $\mathcal{E}_{\mathcal{B}}$ (resulting in $\mathcal{E}'_{\mathcal{B}'}$) to a local change in a BN $\mathcal{B}$ (resulting in $\mathcal{B}'$). The objective of these methods is to make $\mathcal{E}_{\mathcal{B}}$ valid for the new BN. In addition to the incremental compilation of ETs, we also propose a strategy to search in the space of EOs given a BN $\mathcal{B}$. The purpose of this procedure is to reduce the width of an ET without modifying $\mathcal{B}$. We use a simple and efficient heuristic to address this problem. In this paper, we use the optimization process to refine (reduce the width) of the ETs returned by Algorithms 2–3.

Algorithm 4 swaps the position in $\mathcal{E}_{\mathcal{B}}$ of node X_i with the position of its parent X_p, also changing the parents of any children of X_i in $\mathcal{E}_{\mathcal{B}}$ whose validity is compromised by the swap. Algorithm 4 guarantees the validity of the resulting ETs. Note that after each swap only the clusters of X_i and X_p may change.

The sets of EOs represented by $\mathcal{E}_{\mathcal{B}}$ and the new ET $\mathcal{E}'_{\mathcal{B}}$, which we refer to as $\mathcal{S}$ and $\mathcal{S}'$ respectively, are disjoint (see Fig. 11). In $\mathcal{E}_{\mathcal{B}}$, $(X_p < X_i)_\pi$ for any EO $\pi \in \mathcal{S}$, whereas $(X_i < X_p)_{\pi'}$ in $\mathcal{E}'_{\mathcal{B}}$ for any EO $\pi' \in \mathcal{S}'$.

Algorithm 4 proceeds as follows: First, a copy $\mathcal{E}'_{\mathcal{B}}$ of $\mathcal{E}_{\mathcal{B}}$ is created (line 1), and X_j is set to the parent of X_i in $\mathcal{E}_{\mathcal{B}}$ (line 2). Then, the positions of X_i and X_j are swapped in $\mathcal{E}'_{\mathcal{B}}$ (lines 3 and 4). After that, the children of X_i whose cluster contained X_j set their parent in $\mathcal{E}'_{\mathcal{B}}$ to X_j (lines 5–9), because otherwise these nodes would not be sound.

Lemma 3. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET that represents $\mathcal{B}$ over $\mathcal{X}$. Given $X_i \in \mathcal{X}$, the ET $\mathcal{E}'_{\mathcal{B}}$ representing $\mathcal{B}'$ yielded after applying swap($\mathcal{E}_{\mathcal{B}}, X_i$) in Algorithm 4, is also valid.*

Proof. See Appendix A. □

Fig. 12 shows the result of applying Algorithm 4 to an ET $\mathcal{E}_{\mathcal{B}}$. In this example, the positions of X_3 and X_1 (parent of X_3 in $\mathcal{E}_{\mathcal{B}}$) in the resulting ET $\mathcal{E}_{\mathcal{B}'}$ are swapped. $\mathcal{E}_{\mathcal{B}}$ represents the EOs π of $\{X_1, X_2, X_3, X_4\}$ where $(X_2 < X_1)_\pi$ and $(X_3 < X_1)_\pi$ (e.g., $(X_2, X_3, X_1, X_4), (X_4, X_3, X_2, X_1), \ldots$), while $\mathcal{E}_{\mathcal{B}'}$ represents the EOs π' of $\{X_1, X_2, X_3, X_4\}$ where $(X_2 < X_1)_{\pi'}$ and $(X_1 < X_3)_{\pi'}$ (e.g., $(X_4, X_2, X_1, X_3), (X_2, X_4, X_1, X_3), \ldots$).

We use a greedy heuristic (see Algorithm 5), which, given an ET $\mathcal{E}_{\mathcal{B}}$ and a set of nodes for optimization ($\mathcal{X}_{\mathrm{opt}}$), visits each node $X_i \in \mathcal{X}_{\mathrm{opt}}$ from the shallowest to the deepest, checking at each step whether swapping the position of X_i and its parent reduces the width of the ET.

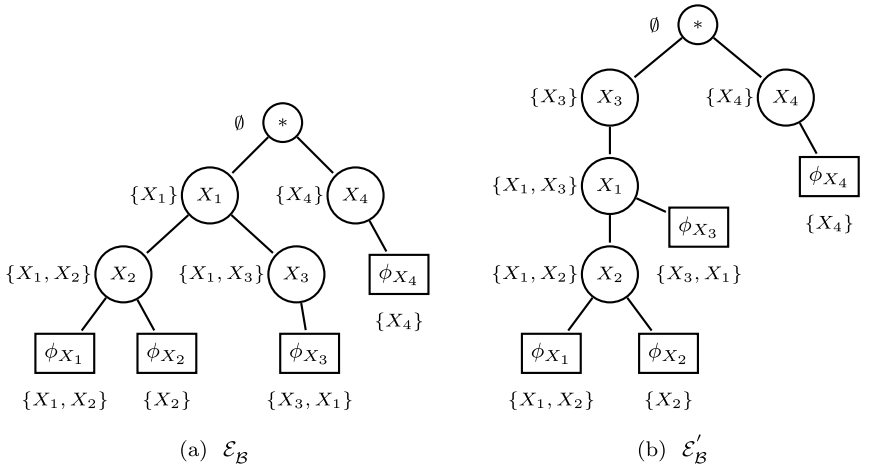


Fig. 12. (a) an ET $\mathcal{E}_{\mathcal{B}}$, and (b) an ET $\mathcal{E}'_{\mathcal{B}}$ yielded after swapping the positions of X_3 and X_1 in $\mathcal{E}_{\mathcal{B}}$.

Input: Valid ET $\mathcal{E}_{\mathcal{B}}$, set of nodes $\mathcal{X}_{\text{opt}}$
Output: Valid ET $\mathcal{E}'_{\mathcal{B}}$
Let $\mathcal{E}'_{\mathcal{B}}$ be a copy of $\mathcal{E}_{\mathcal{B}}$;
Let $\boldsymbol{X}_{\text{opt}}$ be a list that contains the nodes in $\mathcal{X}_{\text{opt}}$ ordered from the shallowest to the deepest;
for $X_i \in \boldsymbol{X}_{\text{opt}}$ **do**
  flag $\leftarrow$ **true**;
  while flag $=$ **true do**
    $\mathcal{E}^1_{\mathcal{B}} \leftarrow \text{swap}(\mathcal{E}'_{\mathcal{B}}, X_i)$;
    if $width(\mathcal{E}^1_{\mathcal{B}}) \leq width(\mathcal{E}'_{\mathcal{B}})$ **then**
      $\mathcal{E}'_{\mathcal{B}} \leftarrow \mathcal{E}^1_{\mathcal{B}}$;
    else
      flag $\leftarrow$ **false**;
    end
  end
end
return $\mathcal{E}'_{\mathcal{B}}$;

Algorithm 5: Optimization of an ET (optimize($\mathcal{E}_{\mathcal{B}}, \mathcal{X}_{\text{opt}}$)).

Orderings that are good for one BN $\mathcal{B}^0$ (i.e., their width is close to the treewidth of $\mathcal{B}^0$) may not be good for the BN $\mathcal{B}$ yielded after applying a local change in $\mathcal{B}^0$. We perform the optimization process after the compilation of each local change. For efficiency, we select the nodes that may have a different cluster after compiling the local change to initialize $\mathcal{X}_{\text{opt}}$, given that they are more likely to produce relevant changes in the width of the ET. Next, we show the set of nodes selected for optimization ($\mathcal{X}_{\text{opt}}$) after compiling arc additions and removals. Each bullet point describes the assignment to $\mathcal{X}_{\text{opt}}$ at a possible scenario. We also explain the reason why the cluster of any of the nodes in $\mathcal{X}_{\text{opt}}$ may have changed.

- Addition of arc $X_{\text{out}} \rightarrow X_{\text{in}}$:
  - If $X_{\text{out}} \in \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}})$, $\mathcal{X}_{\text{opt}} = \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \cap \mathbf{Desc}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}})$:
    The width of the clusters in $\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \cap \mathbf{Desc}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}})$ grows, given that they now contain X_{out}.
  - Else, if $\text{Pa}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \in \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}})$, $\mathcal{X}_{\text{opt}} = (\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}}) \cup \{X_{\text{out}}\}) \cap \mathbf{Desc}_{\mathcal{E}_{\mathcal{B}}}(\text{Pa}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}))$:
    The width of the clusters in $(\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}}) \cup \{X_{\text{out}}\}) \cap \mathbf{Desc}_{\mathcal{E}_{\mathcal{B}}}(\text{Pa}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}))$ may grow, given that they now contain $\mathbf{Dom}(\phi_{X_{\text{in}}})$.
  - Otherwise, $\mathcal{X}_{\text{opt}} = (\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \cup \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}}) \cup \{X_{\text{out}}\}) \setminus (\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \cap \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}}))$:
    Node $\phi_{X_{\text{in}}}$ is set as a descendant of X_{out}, and the nodes in $\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \setminus (\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \cap \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}}))$ are either predecessors or descendants of X_{out} and $\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}}) \setminus (\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \cap \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}}))$ in the new ET.
- Removal of arc $X_{\text{out}} \rightarrow X_{\text{in}}$, $\mathcal{X}_{\text{opt}} = \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \cap \mathbf{Desc}_{\mathcal{E}_{\mathcal{B}}}(X_j)$:
  Let X_j be the last node that had a new child X_i assigned by Algorithm 3. The nodes in $\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \cap \mathbf{Desc}_{\mathcal{E}_{\mathcal{B}}}(X_j)$ may have smaller clusters in the new ET.

The optimization of an ET takes polynomial time in the number of variables and in the width of the ET (see Theorem 2).

4.3. Learning elimination trees from data

Using the incremental compilation and optimization methods described above, it is rather straightforward to learn ETs from a dataset $\mathcal{D}$ in combination with any score+search BN learning method that applies local changes during the search. Learning low inference complexity BNs with this approach is also easily derived. It can be achieved by bounding the width of each ET during the learning process (which bounds the treewidth of their corresponding BNs).

Theorem 1 ensures that any algorithm that uses the above strategy will always produce valid ETs.

Theorem 1. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET over $\mathcal{X}$, and $\mathcal{E}'_{\mathcal{B}'}$ the result of incrementally compiling on $\mathcal{E}_{\mathcal{B}}$ any local change in $\mathcal{B}$ using Algorithms 2 and 3 and optimizing the resulting ET using Algorithm 5. Then $\mathcal{E}'_{\mathcal{B}'}$ is a valid ET.*

Proof. See Appendix A. □

As we apply the incremental compilation and optimization methods to each candidate during the learning process, efficiency is a critical issue. Theorem 2 bounds the computational time complexity of the incremental compilation and optimization methods proposed above.

Theorem 2. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET over a set of variables $\mathcal{X} = \{X_1, \ldots, X_n\}$. The process described in Theorem 1 to output $\mathcal{E}'_{\mathcal{B}'}$ can be performed in time $O(n^2 \cdot width(\mathcal{E}_{\mathcal{B}}))$.*

Proof. See Appendix B. □

Let $\mathcal{A}$ be any algorithm that learns the structure of a BN using only local changes in the structure of the network during the learning process. $\mathcal{A}$ can be adapted to learn low inference complexity BNs compiling and optimizing (Algorithms 2–5) all the local changes that are applied to the BN $\mathcal{B}$ to its respective ET $\mathcal{E}_{\mathcal{B}}$. Thus, $\mathcal{E}_{\mathcal{B}}$ can be used to bound the treewidth of $\mathcal{B}$ using Definition 11. Note that the input for the adaptation of $\mathcal{A}$ should be an ET $\mathcal{E}^0_{\mathcal{B}0}$ valid for the initial BN $\mathcal{B}^0$.

5. Experimental results

In this section, we empirically analyzed the performance of the proposed framework in terms of fitting and computational complexity. Although our approach could be used with most score+search BN learning methods, in the experiments we combine the incremental compilation and optimization methods proposed in Section 4 with a greedy hill-climbing for the structure search. We call the resulting method hc-ET. We compared hc-ET with k-greedy and k-MAX to highlight the advantages and drawbacks of using our approach to learn bounded treewidth BNs. We also tested a polynomial version of hc-ET that only considers arc additions during the structure search. We call this method hc-ET-poly.

To perform the experiments, we used 22 real-world datasets. These datasets were previously used in several papers [53,62–64], and can be found at https://github.com/UCLA-StarAI/Density-Estimation-Datasets. Additionally, we generated synthetic data from 12 real-world BNs. These BNs were obtained from the bnlearn BN repository http://www.bnlearn.com/bnrepository/, and are cited therein. Table 1 briefly describes the basic properties of each dataset.

For each dataset we learned three BNs with each of the compared methods, using different treewidth bounds (3, 5 and 7). In all cases, the score function to maximize was BIC. k-greedy and k-MAX require to fix a maximum execution time, which we set to n seconds (i.e., a second for each variable) to compute the cache of best parent sets and $n/10$ seconds for the structure search. These values were used by Scanagatta et al. [53] in their experiments. To compare the results we used the following performance measures: the BIC score and the log-likelihood (LL) of the models in the training dataset, the learning time, and the treewidth of the returned models.

We analyzed the significance of the differences found for each performance measure in all the datasets and for all the treewidth bounds using the Friedman test with $\alpha = 0.05$ and Holm's [65] and Shaffer's [66] post-hoc procedures. Both Holm's and Shaffer's procedures associate pairwise comparisons with a set of hypotheses and perform a step-down process with the corresponding set of ordered p-values to adjust the value of α [67].

Experiments were performed on a computer with an Intel Core i7-6700K CPU at 4.00 GHz with 16 GB main memory, running Ubuntu 16.04 LTS. hc-ET and hc-ET-poly were written in Python 2.7.12 and C^{++}11 (version 5.4.0), while k-greedy and k-MAX were downloaded from http://ipg.idsia.ch/software/blip and are written in Java.

5.1. Comparison

Tables 2–4 give an overview of the results obtained using the treewidth bounds 3, 5 and 7, respectively. For each performance measure, the mean rank ± the standard deviation of each method over all the datasets is shown. The ranking of the methods is given by their average performance (BIC, LL and time) compared to the rest (i.e., the best is ranked the first and the worst is ranked the fourth). The mean treewidth ± the standard deviation is also shown. The detailed results are supplied as Supplementary Material.

Table 1
Basic properties of the datasets.

(a) Real-world datasets

Dataset	N. vars	N. inst
NLTCS	16	16,181
MSNBC1	17	291,326
KDDCup	65	180,092
Plants	69	17,412
Audio	100	15,000
Jester	100	9,000
Netflix	100	15,000
Accidents	111	12,758
Retail	135	22,041
Pumsb-star	163	12,262
DNA	180	1,600
Kosarek	190	33,375
MSWeb	294	29,441
Book	500	8,700
EachMovie	500	4,525
WebKB	839	2,803
Reuters-521	889	6,532
20 NewsGroup	910	11,293
Movie reviews	1,001	1,600
BBC	1,058	1,670
Voting	1,359	1,214
Ad	1,556	2,461

(b) Synthetic datasets

Dataset	N. vars	N. inst
Hailfinder	56	5,000
Hepar II	70	5,000
Win95pts	76	5,000
Pathfinder	135	5,000
Munin1	186	5,000
Andes	223	5,000
Diabetes	413	5,000
Pigs	413	5,000
Link	724	5,000
Munin2	1,003	5,000
Munin3	1,041	5,000
Munin4	1,038	5,000

Table 2
Comparison of the methods in all the datasets, using a treewidth bound of 3. The optimal results are denoted in boldface.

	hc-ET	hc-ET-poly	k-greedy	k-MAX
Mean rank BIC	**1.32 ± 0.53**	1.79 ± 0.54	3.79 ± 0.41	3.09 ± 0.62
Mean rank LL	**1.38 ± 0.55**	1.68 ± 0.47	3.79 ± 0.41	3.15 ± 0.56
Mean rank time	1.88 ± 0.33	**1.12 ± 0.33**	3.56 ± 0.5	3.44 ± 0.5
Mean treewidth	3 ± 0	3 ± 0	2.85 ± 0.66	2.68 ± 0.77

Table 3
Comparison of the methods in all the datasets, using a treewidth bound of 5. The optimal results are denoted in boldface.

	hc-ET	hc-ET-poly	k-greedy	k-MAX
Mean rank BIC	**1.35 ± 0.69**	1.91 ± 0.57	3.76 ± 0.61	2.97 ± 0.63
Mean rank LL	**1.47 ± 0.71**	1.71 ± 0.52	3.71 ± 0.63	3.12 ± 0.59
Mean rank time	1.88 ± 0.33	**1.12 ± 0.33**	3.56 ± 0.5	3.44 ± 0.5
Mean treewidth	4.88 ± 0.48	4.91 ± 0.38	3.56 ± 1.13	3.41 ± 1.08

Table 4
Comparison of the methods in all the datasets, using a treewidth bound of 7. The optimal results are denoted in boldface.

	hc-ET	hc-ET-poly	k-greedy	k-MAX
Mean rank BIC	**1.35 ± 0.69**	1.97 ± 0.58	3.79 ± 0.59	2.88 ± 0.69
Mean rank LL	**1.29 ± 0.46**	1.82 ± 0.52	3.74 ± 0.62	3.15 ± 0.56
Mean rank time	1.91 ± 0.29	**1.09 ± 0.29**	3.41 ± 0.5	3.59 ± 0.5
Mean treewidth	6.74 ± 0.96	6.74 ± 0.86	3.91 ± 1.31	4 ± 1.44

Figs. 13–15 graphically present the results obtained with Holm's and Shaffer's procedure for each performance measure in all datasets. In the figures, groups of methods that are not significantly different are connected with a thick horizontal line. We used the graphical representation proposed by Demšar [68]. Each figure represents both procedures, given that the significant differences observed by Shaffer's procedure are identical to those observed by Holm's procedure.

Fig. 13 shows significant differences between the BIC score achieved by all the methods; hc-ET performs the best overall, followed by hc-ET-poly, k-MAX, and k-greedy. Moreover, Tables 2–4 show that similar results can be found for all the tested treewidth bounds. The detailed results also show that hc-ET performs better than k-MAX and k-greedy in over 94% of the experiments, and performs better than hc-ET-poly in around 80% of the experiments. The treewidth of the models output by each method suggests that one of the reasons why our proposal manages to optimize better the BIC score is that it allows a tighter fitting to the treewidth bound.

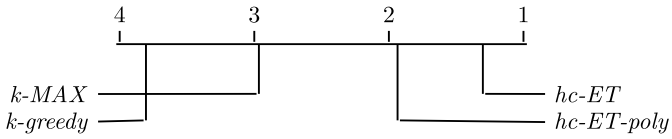


Fig. 13. Comparison of BIC scores with Holm's and Shaffer's tests.

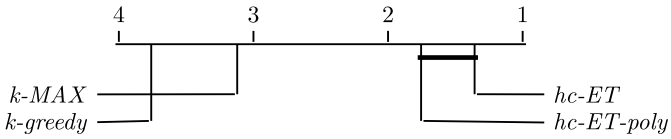


Fig. 14. Comparison of log-likelihood with Holm's and Shaffer's tests.

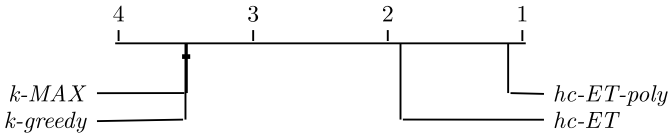


Fig. 15. Comparison of learning time with Holm's and Shaffer's tests.

The comparison of the log-likelihood in Tables 2–4 and Fig. 14 leads us to conclusions that are similar to those drawn for the BIC score. Nevertheless, in this case no significant differences were found between hc-ET and hc-ET-poly. This suggests that hc-ET-poly requires a higher number of parameters to obtain a similar fitting to hc-ET in terms of log-likelihood.

As shown in Fig. 15, we observed significant differences between the learning time of all the methods with the exception of k-MAX and k-greedy. The latter result was expected, given that the imposed limit in execution time of both approaches is the same. hc-ET-poly is the fastest method in all cases, followed by hc-et. However, we must be cautious when interpreting these results. First, these methods are implemented in different programming languages. Second, the bound in execution time set for k-greedy and k-MAX compels their learning time to scale linearly. Therefore, the difference in learning time between our method and k-MAX and k-greedy is clearly higher in the smaller datasets. Finally, although hc-et takes slightly more time than hc-ET-poly in all the experiments, we think that the improvement in BIC score is worthwhile in most situations.

6. Conclusions and future research

Traditional methods for learning BNs usually output models where exact inference is intractable. In this paper, we provide a novel framework for learning tractable BNs. We defined valid ETs, and proposed compilation methods for adapting valid ETs to any local change that may be applied to a BN (i.e., arc addition, removal, and reversal). We proved that the proposed methods always return valid ETs in polynomial time (Theorems 1 and 2). Our approach can be easily combined with any score+search BN learning method that uses only local changes in the network during the structure search. Valid ETs can be used to search in the combined space of DAGs and EOs, avoiding redundant solutions (i.e., all the EOs that are equivalent for a BN are represented by the same valid ET). Hence, we used this representation to efficiently bound the inference complexity of each BN during the learning process.

Experimental results showed that our approach places a tight upper bound on the inference complexity of the networks. The models learned with the proposed methods were competitive with other state-of-the-art methods, performing better in terms of BIC score and log-likelihood in most cases.

Future research will focus on adapting this framework to learning tractable multidimensional BN classifiers (MBCs) [69, 70]. Most probable explanations can be computed in polynomial time in the treewidth of the pruned graph of an MBC [71], that is, a transformation of its structure that entails moralizing the respective DAG and removing the feature variables from the resultant graph. Thus, the application of the methods proposed in this paper to bound the treewidth of the pruned graph of an MBC should lead to tractable MBCs.

Also, we aim to study the relationship between the density of DAGs and the number of equivalent EOs. This would clarify the situations in which it is better to use ETs during the learning process.

Existing methods for learning BNs from incomplete datasets, such as the structural expectation–maximization algorithm [72], require inference during the learning process. We plan to bound the inference complexity of the models to make the structure search tractable when there are missing values or latent variables. Scanagatta et al. [53] successfully introduced the k-MAX algorithm in the maximization step of the structural expectation–maximization algorithm. Thus, we think that adapting our proposal to this problem could lead to promising results.

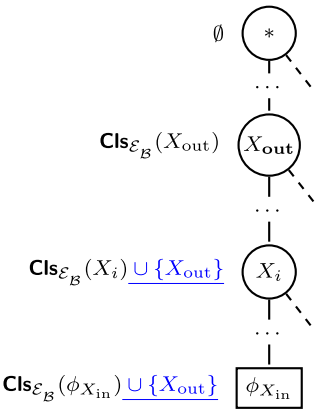


Fig. 16. ET $\mathcal{E}'_{\mathcal{B}'}$ yielded after compiling an arc addition $X_{\text{out}} \rightarrow X_{\text{in}}$ in $\mathcal{E}_{\mathcal{B}}$ when $X_{\text{out}} \in \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}})$. The value of the cluster of each node in $\mathcal{E}'_{\mathcal{B}'}$ is shown near to the respective node, and the changes in the clusters with respect to their value in $\mathcal{E}_{\mathcal{B}}$ are underlined.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science, Innovation and Universities through the Cajal Blue Brain (C080020-09; the Spanish partner of the Blue Brain initiative from EPFL) and TIN2016-79684-P projects, by the Regional Government of Madrid through the S2013/ICE-2845-CASI-CAM-CM project, and by Fundación BBVA grants to Scientific Research Teams in Big Data 2016. This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under Specific Grant Agreement No. 785907 (HBP SGA2). M. Benjumeda is supported by a predoctoral contract for the formation of doctors from the Spanish Ministry of Science, Innovation and Universities (BES-2014-068637).

Appendix A. Proof of Theorem 1

We use the following lemmas to prove that the compilation and optimization methods proposed in this paper always return valid ETs.

Lemma 1. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET that represents $\mathcal{B}$ over $\mathcal{X}$. Given $X_{out}, X_{in} \in \mathcal{X}$, the ET $\mathcal{E}'_{\mathcal{B}'}$ yielded after applying add($\mathcal{E}_{\mathcal{B}}, X_{out}, X_{in}$) in Algorithm 2, is also a valid ET representing $\mathcal{B}'$ over $\mathcal{X}$.*

Proof. By cases:

We prove that, for each possible arc addition scenario, Algorithm 2 always outputs valid ETs. We show that, in each case, all the nodes are complete (i.e., for each node its cluster in $\mathcal{E}'_{\mathcal{B}'}$ contains its parents (Definition 9)) and sound (i.e., for each node its cluster in $\mathcal{E}'_{\mathcal{B}'}$ is a subset of its predecessors and itself (Definition 8)).

- *Case 1*: $X_{\text{out}} \in \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}})$.
  This occurs when neither of the conditions in lines 4 and 6 (Algorithm 1) are fulfilled. Algorithm 1 does not produce any change in the structure of $\mathcal{E}_{\mathcal{B}}$ (see Fig. 16). Hence, neither the parents nor the predecessors of each node in $\mathcal{E}_{\mathcal{B}}$ change. For each node $X_i \in (\mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) \cap \mathbf{Desc}_{\mathcal{E}'_{\mathcal{B}'}}(X_{\text{out}})) \cup \{\phi_{X_{\text{in}}}\}$, the cluster of X_i now contains X_{out}, but $X_{\text{out}} \in \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_i)$. Hence, each X_i is sound. As $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}'}}(X_i) \supseteq \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_i)$, each X_i is complete, and therefore valid. There are no changes in the clusters of the other nodes. Hence, they are valid.
- *Case 2*: $X_f = \text{Pa}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \in \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}})$ (line 4 of Algorithm 2).
  Here, Algorithm 1 sets $\text{Pa}_{\mathcal{E}'_{\mathcal{B}'}}(\phi_{X_{\text{in}}})$ to X_{out} (line 5), and the predecessors and parents of the other nodes are unchanged (see Fig. 17).
  - For each node $X_i \in \mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) \cap \mathbf{Desc}_{\mathcal{E}'_{\mathcal{B}'}}(X_f)$, we have that $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}'}}(X_i) = \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_i) \cup \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}})$. First, $\mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_i) \subseteq \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_i) = \mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}'}}(X_i)$. Also, $\mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \subseteq \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) = \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_f) \cup \{X_f\} \subseteq \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_i) = \mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}'}}(X_i)$. Therefore, $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}'}}(X_i) \subseteq \mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}'}}(X_i)$, and X_i is sound in $\mathcal{E}'_{\mathcal{B}'}$. As $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) = \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \cup \{X_{\text{out}}\}$ and $\mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) \supseteq \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}}) \cup \{X_{\text{out}}\}$, $\phi_{X_{\text{in}}}$ is sound. The rest of the nodes are sound given that there are no changes in their clusters.
  - Node $\phi_{X_{\text{in}}}$ is complete given that $X_{\text{out}} = \text{Pa}_{\mathcal{E}'_{\mathcal{B}'}}(\phi_{X_{\text{in}}})$ and $X_{\text{out}} \in \mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}'}}(\phi_{X_{\text{in}}})$. The rest of the nodes are complete given that $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}'}}(X_i) \supseteq \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_i)$ and $\text{Pa}_{\mathcal{E}'_{\mathcal{B}'}}(X_i) = \text{Pa}_{\mathcal{E}_{\mathcal{B}}}(X_i)$ for each $X_i \in (\mathcal{X} \cup \mathbf{Leaves}(\mathcal{E}_{\mathcal{B}})) \setminus \{\phi_{X_{\text{in}}}\}$.

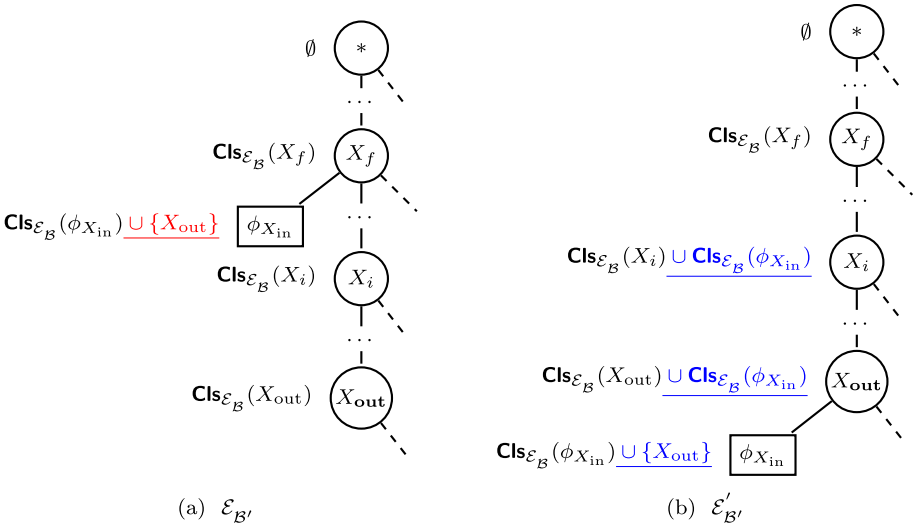


Fig. 17. (a) $\mathcal{E}_{\mathcal{B}'}$ is the ET yielded after adding $X_{\text{out}} \to X_{\text{in}}$ in $\mathcal{B}$ when $X_f \in \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(X_{\text{out}})$ before the compilation of the arc addition; (b) $\mathcal{E}'_{\mathcal{B}'}$ is the result of compiling the arc addition $X_{\text{out}} \to X_{\text{in}}$ in $\mathcal{E}_\mathcal{B}$. The value of the cluster of each node in $\mathcal{E}'_{\mathcal{B}'}$ is shown near to the respective node, and the changes in the clusters with respect to their value in $\mathcal{E}_\mathcal{B}$ are underlined. The changes that compromise the validity of the ET are highlighted in red. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

- *Case 3*: $X_{\text{out}} \notin \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(\phi_{X_{\text{in}}})$ and $X_f \notin \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(X_{\text{out}})$ (line 6 of Algorithm 2).
  In this case, there are two possible output ETs, $\mathcal{E}^1_{\mathcal{B}'}$ and $\mathcal{E}^2_{\mathcal{B}'}$ (line 8).
  1. In $\mathcal{E}^1_{\mathcal{B}'}$ (see Fig. 18b), $\text{Pa}_{\mathcal{E}^1_{\mathcal{B}'}}(X_k)$ is set to X_{out} (line 10):
     - Each node X_i that is not in $(\mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) \cap \mathbf{Desc}_{\mathcal{E}^1_{\mathcal{B}'}}(X_m)) \cup \{\phi_{X_{\text{in}}}\}$ has the same parents and clusters in $\mathcal{E}_\mathcal{B}$ and $\mathcal{E}^1_{\mathcal{B}'}$, and $\mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}'}}(X_i) \supseteq \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(X_i)$. Hence, it is valid.
     - Each node X_i in $\mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}'}}(X_k) \cap \mathbf{Desc}_{\mathcal{E}^1_{\mathcal{B}'}}(X_m)$ has the same predecessors and parents in $\mathcal{E}_\mathcal{B}$ and $\mathcal{E}^1_{\mathcal{B}'}$, and $\mathbf{Cls}_{\mathcal{E}^1_{\mathcal{B}'}}(X_i) = \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_i) \cup \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_k) \setminus \{X_k\}$ (making X_i complete). As $\mathbf{Cls}_{\mathcal{E}^1_{\mathcal{B}'}}(X_k) \setminus \{X_k\} \subseteq \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(X_k) \subseteq \mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}'}}(X_i)$, each X_i is also sound. Thus, each X_i is valid.
     - For each $X_i \in (\mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) \cup \{\phi_{X_{\text{in}}}\}) \cap \mathbf{Desc}_{\mathcal{E}^1_{\mathcal{B}'}}(X_{\text{out}})$, $\mathbf{Cls}_{\mathcal{E}^1_{\mathcal{B}'}}(X_i) = \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_i) \cup \{X_{\text{out}}\}$, $\mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}'}}(X_i) \cup \{X_i\} \supseteq \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_i) \cup \{X_{\text{out}}\}$, $\text{Pa}_{\mathcal{E}^1_{\mathcal{B}'}}(X_i) = \text{Pa}_{\mathcal{E}_\mathcal{B}}(X_i)$ if $X_i \neq X_k$, and $\text{Pa}_{\mathcal{E}^1_{\mathcal{B}'}}(X_k) = X_{\text{out}}$. Hence, X_i is valid.

     As each node in $\mathcal{E}^1_{\mathcal{B}'}$ is valid, $\mathcal{E}^1_{\mathcal{B}'}$ is valid.
  2. In $\mathcal{E}^2_{\mathcal{B}'}$ (see Fig. 18c), $\text{Pa}_{\mathcal{E}^1_{\mathcal{B}'}}(X_h)$ is set to X_f (line 12).
     - Each node has the same parent in $\mathcal{E}^2_{\mathcal{B}'}$ and $\mathcal{E}_\mathcal{B}$, with the exception of $\phi_{X_{\text{in}}}$ and X_h, where $\text{Pa}_{\mathcal{E}^2_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) = X_{\text{out}}$ (line 13) and $\text{Pa}_{\mathcal{E}^2_{\mathcal{B}'}}(X_h) = X_f$. All the nodes are complete, given that $X_{\text{out}} \in \mathbf{Cls}_{\mathcal{E}^2_{\mathcal{B}'}}(\phi_{X_{\text{in}}})$, $\mathbf{Cls}_{\mathcal{E}^2_{\mathcal{B}'}}(X_h) \supseteq \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(\phi_{X_{\text{in}}}) \supseteq \{X_f\}$, and for each other node X_i, $\mathbf{Cls}_{\mathcal{E}^2_{\mathcal{B}'}}(X_i) \supseteq \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_i)$.
     - For each node X_i not in $(\mathbf{Pred}_{\mathcal{E}^2_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) \cap \mathbf{Desc}_{\mathcal{E}^2_{\mathcal{B}'}}(X_m)) \cup \{\phi_{X_{\text{in}}}\}$, the clusters of X_i are the same in $\mathcal{E}^2_{\mathcal{B}'}$ and in $\mathcal{E}_\mathcal{B}$ and $\mathbf{Pred}_{\mathcal{E}^2_{\mathcal{B}'}}(X_i) \supseteq \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(X_i)$. Hence each X_i is sound.
     - For each $X_i \in \mathbf{Pred}_{\mathcal{E}^2_{\mathcal{B}'}}(X_h) \cap \mathbf{Desc}_{\mathcal{E}^2_{\mathcal{B}'}}(X_m)$, $\mathbf{Cls}_{\mathcal{E}^2_{\mathcal{B}'}}(X_i) = \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_i) \cup \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_h) \setminus \{X_h\}$ and $\mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_h) \setminus \{X_h\} \subseteq \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(X_h) \subseteq \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(X_i) = \mathbf{Pred}_{\mathcal{E}^2_{\mathcal{B}'}}(X_i)$. Thus, each X_i is sound.
     - For each $X_i \in \mathbf{Pred}_{\mathcal{E}^2_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) \cap \mathbf{Desc}_{\mathcal{E}^2_{\mathcal{B}'}}(X_f)$, $\mathbf{Cls}_{\mathcal{E}^2_{\mathcal{B}'}}(X_i) = \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_i) \cup \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(\phi_{X_{\text{in}}})$ and $\mathbf{Pred}_{\mathcal{E}^2_{\mathcal{B}'}}(X_i) = \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(X_i) \cup \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(\phi_{X_{\text{in}}}) \supseteq \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(X_i) \setminus \{X_i\} \cup \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(\phi_{X_{\text{in}}})$. Hence, X_i is sound.
     - Node $\phi_{X_{\text{in}}}$ contains X_{out} in its cluster and predecessors in $\mathcal{E}^2_{\mathcal{B}'}$. Hence, $\mathbf{Cls}_{\mathcal{E}^2_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) = \mathbf{Cls}_{\mathcal{E}_\mathcal{B}}(\phi_{X_{\text{in}}}) \cup \{X_{\text{out}}\} \subseteq \mathbf{Pred}_{\mathcal{E}_\mathcal{B}}(\phi_{X_{\text{in}}}) \cup \{X_{\text{out}}\} \subseteq \mathbf{Pred}_{\mathcal{E}^2_{\mathcal{B}'}}(\phi_{X_{\text{in}}})$, making $\phi_{X_{\text{in}}}$ sound.

     As every node in $\mathcal{E}^2_{\mathcal{B}'}$ is sound and complete, $\mathcal{E}^2_{\mathcal{B}'}$ is valid. □

□

Lemma 2. *Let $\mathcal{E}_\mathcal{B}$ be a valid ET that represents $\mathcal{B}$ over $\mathcal{X}$. Given $X_{out}, X_{in} \in \mathcal{X}$, the ET $\mathcal{E}'_{\mathcal{B}'}$ that represents $\mathcal{B}'$ output after applying remove($\mathcal{E}_\mathcal{B}$, X_{out}, X_{in}) in Algorithm 3, is also valid.*

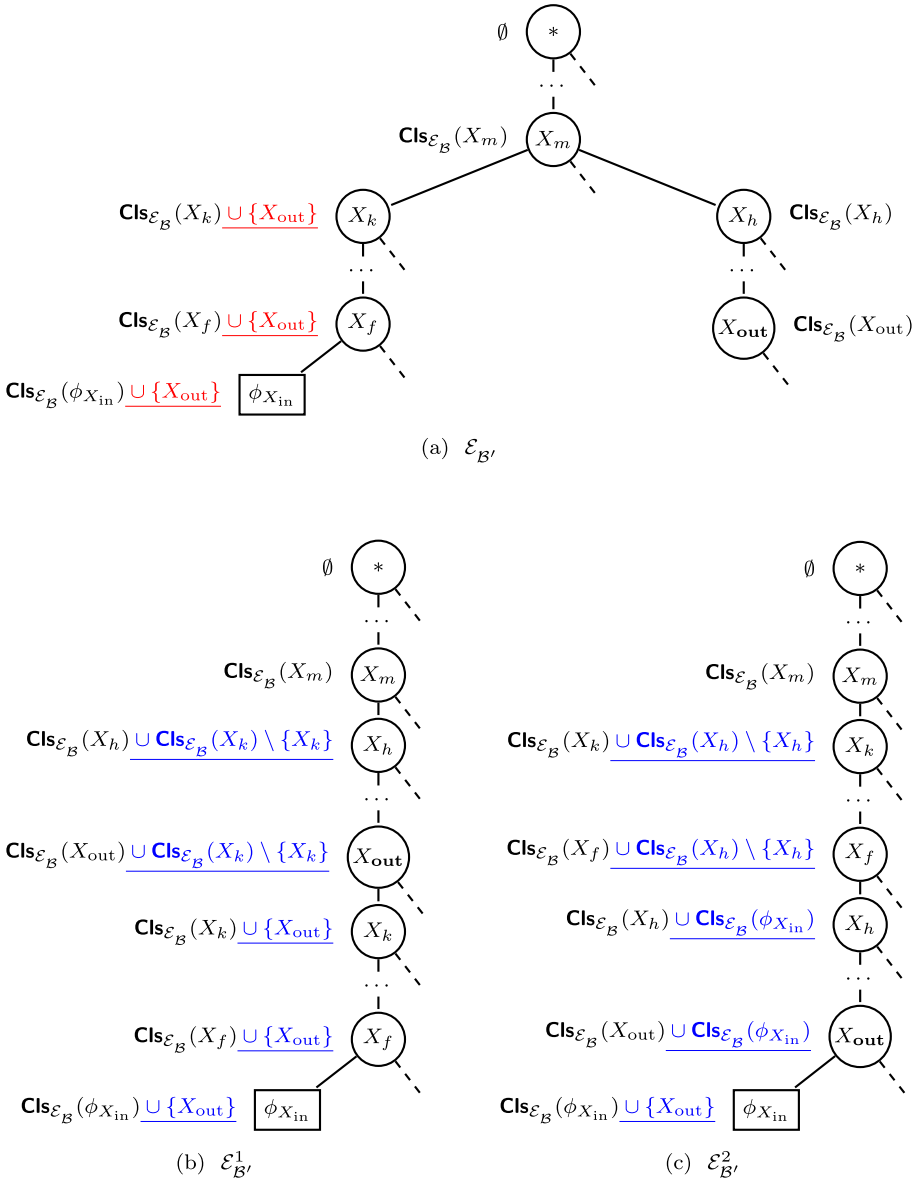


Fig. 18. $\mathcal{E}_{\mathcal{B}'}$ (a) is the ET yielded after adding $X_{\text{out}} \rightarrow X_{\text{in}}$ in $\mathcal{B}$ when $X_{\text{out}} \notin \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(\phi_{X_{\text{in}}})$ and $X_f \notin \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_{\text{out}})$ before the compilation of the arc addition. $\mathcal{E}^1_{\mathcal{B}'}$ (b) and $\mathcal{E}^2_{\mathcal{B}'}$ (c) correspond to the two possible outcomes of compiling the arc addition. The value of the cluster of each node in $\mathcal{E}'_{\mathcal{B}'}$ is shown near to the respective node, and the changes in the clusters with respect to their value in $\mathcal{E}_{\mathcal{B}}$ are underlined. The changes that compromise the validity of the ET are highlighted in red.

Proof. By induction. We show that after removing an arc from $\mathcal{E}_{\mathcal{B}}$ only one node X_i may not be complete (base case). In each iteration *iter* of Algorithm 3, the completeness of X_i is amended and $\mathcal{E}^{iter}_{\mathcal{B}'}$ is built, and only one other node X'_i, which was a predecessor of X_i in the previous ET, may not be complete after the change. It is evident that eventually node X'_i will be complete (e.g., when the parent of X'_i is the root node $*$).

Base case:

Given a valid ET $\mathcal{E}_{\mathcal{B}}$, removing arc $X_{\text{out}} \rightarrow X_{\text{in}}$ from $\mathcal{B}$ will produce an ET $\mathcal{E}_{\mathcal{B}'}$. For each $X_h \in \mathcal{X} \cup \mathbf{Leaves}(\mathcal{E}_{\mathcal{B}'})$, $\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}'}}(X_h) = \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_h)$, $\text{Pa}_{\mathcal{E}_{\mathcal{B}'}}(X_h) = \text{Pa}_{\mathcal{E}_{\mathcal{B}}}(X_h)$, $\mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_h) \supseteq \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}'}}(X_h) \supseteq \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_h) \setminus \{X_{\text{out}}\}$ if $X_h \in \mathbf{Desc}_{\mathcal{E}_{\mathcal{B}'}}(X_{\text{out}}) \cap \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) \cup \{\phi_{X_{\text{in}}}\}$, and $\mathbf{Cls}_{\mathcal{E}_{\mathcal{B}'}}(X_h) = \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_h)$ otherwise. Therefore, each node in $\mathcal{E}_{\mathcal{B}'}$ is sound, and only one node X_i such that $X_i \in \mathbf{Ch}_{\mathcal{E}_{\mathcal{B}'}}(X_{\text{out}}) \cap (\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}'}}(\phi_{X_{\text{in}}}) \cup \{\phi_{X_{\text{in}}}\})$ may not be complete.

Iterative step:

Assume that $\mathcal{E}^1_{\mathcal{B}'}$ (Fig. 19a) is sound and only node X_i is not complete. Algorithm 3 sets $\text{Pa}_{\mathcal{E}^1_{\mathcal{B}'}}(X_i) = X_j$ (line 7), that is, the deepest node in $\mathcal{E}^1_{\mathcal{B}'}$ belonging to $\mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_i) \setminus \{X_i\}$ (line 6). Hence, all X_i and their descendants are sound. Thus, node

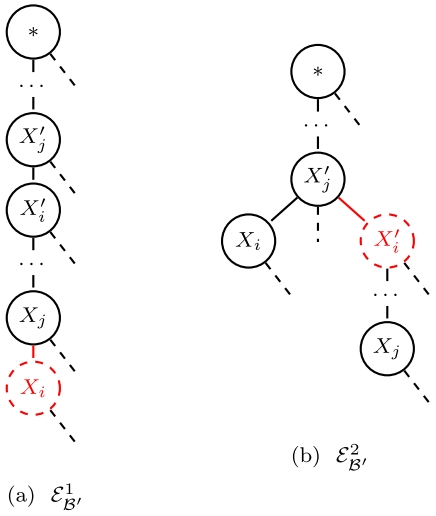


Fig. 19. (a) $\mathcal{E}^1_{\mathcal{B}'}$ is the ET visited at an iterative step of compiling an arc removal; (b) $\mathcal{E}^2_{\mathcal{B}'}$ is the result of performing this iterative step.

X_i is complete in the resulting ET $\mathcal{E}^2_{\mathcal{B}'}$ (Fig. 19b). Node $X'_i = \mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}'}}(X_i) \cap \mathbf{Ch}_{\mathcal{E}^1_{\mathcal{B}'}}(X_j)$ may not be complete in $\mathcal{E}^2_{\mathcal{B}'}$. For each node $X_h \in \mathbf{Pred}_{\mathcal{E}^1_{\mathcal{B}'}}(X_i) \cap \mathbf{Desc}_{\mathcal{E}^1_{\mathcal{B}'}}(X'_i)$, $\mathbf{Cls}_{\mathcal{E}^2_{\mathcal{B}'}}(X_h) \subseteq \mathbf{Cls}_{\mathcal{E}^1_{\mathcal{B}'}}(X_h) \setminus \mathbf{Cls}_{\mathcal{E}^1_{\mathcal{B}'}}(X_i)$ and $\mathrm{Pa}_{\mathcal{E}^1_{\mathcal{B}'}}(X_h) \notin \mathbf{Cls}_{\mathcal{E}^1_{\mathcal{B}'}}(X_i)$. Hence, each X_h is complete in $\mathcal{E}^2_{\mathcal{B}'}$. The other nodes have the same clusters in $\mathcal{E}^1_{\mathcal{B}'}$ and in $\mathcal{E}^2_{\mathcal{B}'}$. Hence, they are complete. □

Lemma 3. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET that represents $\mathcal{B}$ over $\mathcal{X}$. Given $X_i \in \mathcal{X}$, the ET $\mathcal{E}'_{\mathcal{B}}$ representing $\mathcal{B}'$ yielded after applying swap($\mathcal{E}_{\mathcal{B}}$, X_i) in Algorithm 4, is also valid.*

Proof. Let X_p be the parent of X_i in $\mathcal{E}_{\mathcal{B}}$ (Fig. 20a). Next, we prove that each node in $\mathcal{E}'_{\mathcal{B}}$ (Fig. 20b) is valid after the swap:

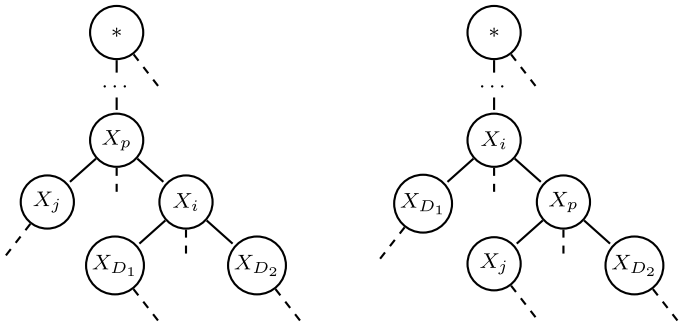
- Each node X_j in $(\mathbf{Desc}_{\mathcal{E}_{\mathcal{B}}}(X_p) \setminus (\mathbf{Desc}_{\mathcal{E}_{\mathcal{B}}}(X_i) \cup \{X_i\})) \cup \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_p)$ has the same parent and clusters in $\mathcal{E}'_{\mathcal{B}}$, and $\mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}}}(X_j) \supset \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_j)$. Hence, each X_j is valid in $\mathcal{E}'_{\mathcal{B}}$.
- Let us divide the descendants of X_i in $\mathcal{E}_{\mathcal{B}}$ into two subsets D_1 and D_2. Let C_1 be the children of X_i in $\mathcal{E}_{\mathcal{B}}$ that do not contain X_p in its cluster. We use D_1 to refer to the nodes in $\mathbf{Desc}_{\mathcal{E}_{\mathcal{B}}}(X_i)$ such that for each $X_j \in D_1$, $(\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_j) \cup \{X_j\}) \cap C_1 \neq \emptyset$, and D_2 to refer to the nodes in $\mathbf{Desc}_{\mathcal{E}_{\mathcal{B}}}(X_i) \setminus D_1$.
  Each node X_j in D_1 has the same parent and cluster in $\mathcal{E}_{\mathcal{B}}$ and $\mathcal{E}'_{\mathcal{B}}$, and $\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_j) = \mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}}}(X_j) \setminus \{X_p\}$. As $X_p \notin \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_j)$, the respective X_j are valid in $\mathcal{E}'_{\mathcal{B}}$.
  Each X_j in D_2 has the same predecessors and clusters in $\mathcal{E}_{\mathcal{B}}$ and $\mathcal{E}'_{\mathcal{B}}$. If $X_j \notin \mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)$, X_j has the same parent in $\mathcal{E}_{\mathcal{B}}$ and $\mathcal{E}'_{\mathcal{B}}$. Otherwise, $\mathrm{Pa}_{\mathcal{E}'_{\mathcal{B}}}(X_j) = X_p$ and $X_p \in \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_j)$. Thus, each X_j is valid in $\mathcal{E}'_{\mathcal{B}}$.
- $X_i \in \mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}}}(X_p)$, given that there are nodes in D_2 whose cluster contains X_i (otherwise $\mathcal{E}_{\mathcal{B}}$ would not be complete). This means that X_p is complete for $\mathcal{E}'_{\mathcal{B}}$. As $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}}}(X_p) \subseteq \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_p) \cup \{X_i\}$ and $\mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}}}(X_p) = \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_p) \cup \{X_i\}$, X_p is sound, and therefore valid for $\mathcal{E}'_{\mathcal{B}}$.
- As $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}}}(X_i) = (\mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_i) \cup \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_p)) \setminus \{X_p\}$, $\mathbf{Pred}_{\mathcal{E}'_{\mathcal{B}}}(X_i) = (\mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_i) \cup \mathbf{Pred}_{\mathcal{E}_{\mathcal{B}}}(X_p)) \setminus \{X_p\}$, and the parent of X_i in $\mathcal{E}'_{\mathcal{B}}$ is the parent of X_p in $\mathcal{E}_{\mathcal{B}}$, X_i is valid for $\mathcal{E}'_{\mathcal{B}}$.

We have shown that each node in $\mathcal{X} \cup \mathbf{Leaves}(\mathcal{E}'_{\mathcal{B}})$ is valid in $\mathcal{E}'_{\mathcal{B}}$. Thus, $\mathcal{E}'_{\mathcal{B}}$ is valid. □

Finally, Theorem 1 can be proved using the above lemmas.

Theorem 1. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET over $\mathcal{X}$, and $\mathcal{E}'_{\mathcal{B}'}$ the result of incrementally compiling on $\mathcal{E}_{\mathcal{B}}$ any local change in $\mathcal{B}$ using Algorithms 2 and 3 and optimizing the resulting ET using Algorithm 5. Then $\mathcal{E}'_{\mathcal{B}'}$ is a valid ET.*

Proof. By Lemmas 1 and 2 we know that if $\mathcal{E}_{\mathcal{B}}$ is valid, the tree returned after compiling a local change in $\mathcal{E}_{\mathcal{B}}$ is also valid. Hence, if the input for Algorithm 5 is a valid ET, we know, by Lemma 3 (the optimization is composed of a sequence of swaps), that it will also return a valid ET. □



(a) $\mathcal{E}_{\mathcal{B}}$ (b) $\mathcal{E}'_{\mathcal{B}}$

Fig. 20. Swap of X_i and X_p. $\mathcal{E}_{\mathcal{B}}$ (a) and $\mathcal{E}'_{\mathcal{B}}$ (b) correspond to the ETs before and after swapping X_i and X_p, respectively. Note that if $\mathcal{E}_{\mathcal{B}}$ is valid, $\mathcal{E}'_{\mathcal{B}}$ is also valid.

Appendix B. Proof of Theorem 2

The following lemmas are used later to prove Theorem 2. First, we need to know the computational cost of outputting the cluster of a node in an ET.

Lemma 4. *Let $\mathcal{E}_{\mathcal{B}}$ be an ET over $\mathcal{X} = \{X_1, \ldots, X_n\}$. The cluster of a node $X_i \in \mathcal{X}$ can be computed in time $O(|\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)| \cdot width(\mathcal{E}_{\mathcal{B}}))$ given the clusters of the nodes in $\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)$.*

Proof. The cluster of node X_i can be output by computing the union of the clusters of its children in $\mathcal{E}_{\mathcal{B}}$ (Definition 7). The union of sets $\mathcal{S}_1, \ldots, \mathcal{S}_m$ can be computed in time $|\mathcal{S}_1| + \cdots + |\mathcal{S}_m|$. As the size of each cluster in $\mathcal{E}_{\mathcal{B}}$ is less than or equal to $\text{width}(\mathcal{E}_{\mathcal{B}}) + 1$, then $\sum_{X_j \in \mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)} |\mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_j)| \leq \sum_{X_j \in \mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)} (\text{width}(\mathcal{E}_{\mathcal{B}}) + 1) = |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)| \cdot (\text{width}(\mathcal{E}_{\mathcal{B}}) + 1)$. Hence, the cluster of X_i can be output in time $O(|\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)| \cdot \text{width}(\mathcal{E}_{\mathcal{B}}))$. □

Lemma 5. *Let $\mathcal{E}_{\mathcal{B}}$ be an ET over $\mathcal{X} = \{X_1, \ldots, X_n\}$. All the clusters in $\mathcal{E}_{\mathcal{B}}$ can be computed in time $O(n \cdot width(\mathcal{E}_{\mathcal{B}}))$.*

Proof. The cluster of a leaf node is its domain. The cluster of the inner nodes can be output bottom-up such that before computing the cluster of node X_i the cluster of each child of X_i in $\mathcal{E}_{\mathcal{B}}$ is known. From Lemma 4, we know that the cluster of each $X_i \in \mathcal{X}$ can be output in time $O(|\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)| \cdot \text{width}(\mathcal{E}_{\mathcal{B}}))$. Hence, the clusters of all the nodes in $\mathcal{X}$ can be computed in time $O(\sum_{X_i \in \mathcal{X}} |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)| \cdot \text{width}(\mathcal{E}_{\mathcal{B}}))$. As $\sum_{X_i \in \mathcal{X}} |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)| \cdot \text{width}(\mathcal{E}_{\mathcal{B}}) < 2n \cdot \text{width}(\mathcal{E}_{\mathcal{B}})$ (there are n inner nodes with only one parent, of which at least one is a child of the root node $*$, and n edges including inner and leaf nodes), all the clusters of $\mathcal{E}_{\mathcal{B}}$ can be computed in time $O(n \cdot \text{width}(\mathcal{E}_{\mathcal{B}}))$. □

A local change in an ET $\mathcal{E}_{\mathcal{B}}$ produces changes in the clusters of the tree, which have an influence on the computational complexity of Algorithm 5.

Lemma 6. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET, and $\mathcal{E}'_{\mathcal{B}}$ the result of swapping (Algorithm 4) a node X_i and its parent in $\mathcal{E}_{\mathcal{B}}$. Then*

$$width(\mathcal{E}'_{\mathcal{B}}) \leq 2 \cdot width(\mathcal{E}_{\mathcal{B}}).$$

Proof. After swapping X_i and its parent X_p in $\mathcal{E}_{\mathcal{B}}$, only the clusters of X_i and X_p may change. On the one hand, $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}}}(X_p) \subseteq \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_p)$. Hence, the width of X_p does not grow. On the other hand, the width of X_i may grow, but $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}}}(X_i) \subseteq (\mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_i) \cup \mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_p)) \setminus \{X_p\}$. Hence, the width of $\mathbf{Cls}_{\mathcal{E}'_{\mathcal{B}}}(X_i)$ is less than $|\mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_i)| + |\mathbf{Cls}_{\mathcal{E}_{\mathcal{B}}}(X_p)| - 1 \leq 2(\text{width}(\mathcal{E}_{\mathcal{B}})) + 1$. This means that $\text{width}(\mathcal{E}'_{\mathcal{B}}) \leq 2 \cdot \text{width}(\mathcal{E}_{\mathcal{B}})$. □

Next, we bound the time complexity of the compilation and optimization methods.

Lemma 7. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET over $\mathcal{X} = \{X_1, \ldots, X_n\}$. The addition of any arc in $\mathcal{B}$ can be compiled in $\mathcal{E}_{\mathcal{B}}$ in time $O(n^2)$ by Algorithm 2.*

Proof. There are no loops in Algorithm 2, and the only operations that cannot be completed in time $O(1)$ are:

- The intersection performed to compute X_m, X_k and X_h (lines 7, 9 and 11 of Algorithm 2), which can be computed in $O(n)$.
- The width of $\mathcal{E}^1_{\mathcal{B}'}$ and $\mathcal{E}^2_{\mathcal{B}'}$. We need to output first the clusters of $\mathcal{E}^1_{\mathcal{B}'}$ and $\mathcal{E}^2_{\mathcal{B}'}$, which can be obtained in time $O(n \cdot \text{width}(\mathcal{E}_{\mathcal{B}}))$. The width of $\mathcal{E}_{\mathcal{B}}$ is the length of its largest cluster minus one (Definition 11), which takes $O(n)$. The complete process takes $O(n \cdot \text{width}(\mathcal{E}_{\mathcal{B}}) + n) = O(n \cdot \text{width}(\mathcal{E}_{\mathcal{B}})) \leq O(n^2)$.

Therefore, the addition of an arc can be compiled in time $O(n^2)$. □

Lemma 8. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET over $\mathcal{X} = \{X_1, \ldots, X_n\}$. The removal of any arc in $\mathcal{B}$ can be compiled in $\mathcal{E}_{\mathcal{B}}$ in time $O(n^2 \cdot width(\mathcal{E}_{\mathcal{B}}))$ by Algorithm 3.*

Proof. In each iteration, Algorithm 3 checks if a node X_i contains its current parent in its cluster (line 5); else, the new parent of X_i is set to the deepest node X'_j, which appears in the cluster of X_i (line 6). Then, the child of X'_j, which was previously a predecessor of X_i, is set as the new X_i, and X'_j is set as the new X_j for the next iteration (lines 8 and 9 of Algorithm 3). Therefore, node X_i is not visited again in the next iterations. This means that there are fewer than n iterations.

The operations in lines 6–9 of Algorithm 3 can be completed in time $O(n)$.

The clusters of several nodes must be output after each iteration. By Lemma 5, we know that the clusters of all the nodes in $\mathcal{X}$ can be computed in time $O(n \cdot \text{width}(\mathcal{E}_{\mathcal{B}}))$.

As there are fewer than n iterations, Algorithm 3 can be run in time $O(n^2 \cdot \text{width}(\mathcal{E}_{\mathcal{B}}) + n^2) = O(n^2 \cdot \text{width}(\mathcal{E}_{\mathcal{B}}))$. □

Lemma 9. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET over $\mathcal{X} = \{X_1, \ldots, X_n\}$. Swapping node X_i and its parent X_p in $\mathcal{E}_{\mathcal{B}}$ using Algorithm 4 and updating the clusters of $\mathcal{E}_{\mathcal{B}}$ can be completed in time $O(width(\mathcal{E}_{\mathcal{B}})(|\boldsymbol{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)| + |\boldsymbol{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_p)|))$.*

Proof. To swap a node $X_i \in \mathcal{X}$ with its parent in $\mathcal{E}_{\mathcal{B}}$, Algorithm 4 assigns a new parent to X_i and to its previous parent X_p. It also assigns X_p as the new parent of any children of X_i whose cluster contains X_p. This can be completed in time $O(|\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)|)$. Note that we can check if X_p belongs to a cluster in time $O(1)$.

After swapping X_i and X_p, only the clusters of X_i and X_p change. By Lemma 4, we know that this can be computed in $O(|\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)|\text{width}(\mathcal{E}_{\mathcal{B}}) + |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_p)|\text{width}(\mathcal{E}_{\mathcal{B}})) = O(\text{width}(\mathcal{E}_{\mathcal{B}})(|\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_i)| + |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}}(X_p)|))$. □

Lemma 10. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET over $\mathcal{X} = \{X_1, \ldots, X_n\}$. Algorithm 5 can be computed in time $O(n^2 \cdot width(\mathcal{E}_{\mathcal{B}}))$.*

Proof. The input of Algorithm 5 is a list of nodes $\boldsymbol{X}_{\text{opt}} = (X_{l(1)}, \cdots, X_{l(m)})$ for optimization. Assuming that these nodes are ordered from the shallowest to the deepest (i.e. the depth of $X_{l(i)}$ is greater than or equal to the depth of $X_{l(i+1)}$), Algorithm 5 starts swapping $X_{l(1)}$ while the width of the ET does not increase, and then it performs the same process with $X_{l(2)}, \cdots, X_{l(m)}$. Thus, the cost of Algorithm 5 is given by the cost of each swap performed during the optimization. According to Lemma 9, Algorithm 5 can be computed in time

$$O\Big(\sum_{i=1}^{m}\sum_{j=1}^{k_i} \text{width}(\mathcal{E}_{\mathcal{B}}^{i,j-1})(|\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(X_{l(i)})| + |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(\text{Pa}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(X_{l(i)}))|)\Big),$$

where:

- $k_i < n$ is the number of times that node $X_{l(i)}$ is swapped.
- $\mathcal{E}_{\mathcal{B}}^{i,j}$ is the ET obtained after swapping node $X_{l(i)}$ j times after nodes $X_{l(1)}, \ldots, X_{l(i-1)}$ have been optimized.
- $\mathcal{E}_{\mathcal{B}}^{i,0} = \mathcal{E}_{\mathcal{B}}^{i-1,k_i}$ if $\text{Pa}_{\mathcal{E}_{\mathcal{B}}^{i-1,k_i}}(X_{l(i-1)}) = *$ (i.e., swapping node $X_{l(i-1)}$ always reduces the width of the ET candidates until its parent is the root node) and $\mathcal{E}_{\mathcal{B}}^{i,0} = \mathcal{E}_{\mathcal{B}}^{i-1,k_i-1}$ otherwise.
- $\mathcal{E}_{\mathcal{B}}^{1,0} = \mathcal{E}_{\mathcal{B}}$.

When the width of a candidate $\mathcal{E}_{\mathcal{B}}^{i,j}$ is bigger than $\text{width}(\mathcal{E}_{\mathcal{B}})$, $\mathcal{E}_{\mathcal{B}}^{i,j}$ is rejected. Thus, by Lemma 6, $\text{width}(\mathcal{E}_{\mathcal{B}}^{i,j-1}) \leq 2 \cdot \text{width}(\mathcal{E}_{\mathcal{B}})$, and

$$\sum_{i=1}^{m}\sum_{j=1}^{k_i} \text{width}(\mathcal{E}_{\mathcal{B}}^{i,j-1})(|\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(X_{l(i)})| + |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(\text{Pa}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(X_{l(i)}))|)$$

$$\leq 2 \cdot \text{width}(\mathcal{E}_{\mathcal{B}})\Big(\sum_{i=1}^{m}\sum_{j=1}^{k_i} |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(X_{l(i)})| + |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(\text{Pa}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(X_{l(i)}))|\Big).$$

The complexity of Algorithm 5 can be output by counting the number of children of each $X_{l(i)}$ and its parent in each iteration.

In any ET $\mathcal{E}_{\mathcal{B}}^{i,j}$, there are less than $2n$ arcs without counting arcs from the root node. Also, note that after swapping node $X_{l(i)}$ with its parent X_p in an ET $\mathcal{E}_{\mathcal{B}}^{i,j}$, $\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j+1}}(X_p) \supseteq \mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j}}(X_p) \setminus \{X_{l(i)}\}$, $\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j+1}}(X_{l(i)}) \subseteq \mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j}}(X_{l(i)}) \setminus \{X_p\}$ and $\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j+1}}(X_p) \cup \mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j+1}}(X_{l(i)}) = \mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j}}(X_p) \cup \mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j}}(l(i))$. This implies that if a node X_c is the child of a node X_h in any $\mathcal{E}_{\mathcal{B}}^{i,0}, \ldots, \mathcal{E}_{\mathcal{B}}^{i,k_i-1}$, it cannot be the child of another node that is not X_h or $X_{l(i)}$ in $\mathcal{E}_{\mathcal{B}}^{i,0}, \ldots, \mathcal{E}_{\mathcal{B}}^{i,k_i-1}$. Therefore, it is evident that $\sum_{j=1}^{k_i} |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(\mathrm{Pa}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(X_{l(i)}))| < 2n$.

To bound $\sum_{i=1}^{m}\sum_{j=1}^{k_i} |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(X_{l(i)})|$, let us focus on the number of children that each node has when it is swapped. As the nodes in $\mathcal{X}_{\mathrm{opt}}$ are visited from the shallowest to the deepest, if a node X_h is a child of node $X_{l(i)}$ when $X_{l(i)}$ is optimized, it cannot be a child of another node $X_{l(j)} \in \mathcal{X}_{\mathrm{opt}}$ when $X_{l(j)}$ is optimized. Thus, each node X_h can be counted less than n times, and given that there are $2n$ inner and leaf nodes in $\mathcal{E}_{\mathcal{B}}$, $\sum_{i=1}^{m}\sum_{j=1}^{k_i} |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(X_{l(i)})| < 2n^2$.

Finally, $2 \cdot \mathrm{width}(\mathcal{E}_{\mathcal{B}}) \cdot (\sum_{i=1}^{m}\sum_{j=1}^{k_i} |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(X_{l(i)})| + \sum_{i=1}^{m}\sum_{j=1}^{k_i} |\mathbf{Ch}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(\mathrm{Pa}_{\mathcal{E}_{\mathcal{B}}^{i,j-1}}(X_{l(i)}))|) < \mathrm{width}(\mathcal{E}_{\mathcal{B}})(4n^2 + 4n)$. Therefore, Algorithm 5 can be computed in time $O(n^2 \cdot \mathrm{width}(\mathcal{E}_{\mathcal{B}}))$. □

Theorem 2 can be proved using the lemmas shown above.

Theorem 2. *Let $\mathcal{E}_{\mathcal{B}}$ be a valid ET over a set of variables $\mathcal{X} = \{X_1, \ldots, X_n\}$. The process described in Theorem 1 to output $\mathcal{E}'_{\mathcal{B}'}$ can be performed in time $O(n^2 \cdot width(\mathcal{E}_{\mathcal{B}}))$.*

Proof. By Lemmas 7, 8 and 10, we know that both the compilation and optimization process can be performed in time $O(n^2 \cdot \mathrm{width}(\mathcal{E}_{\mathcal{B}}))$. □

Appendix C. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.artint.2018.11.007.

References

[1] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, 1988.
[2] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques – Adaptive Computation and Machine Learning, The MIT Press, 2009.
[3] C. Bielza, P. Larrañaga, Discrete Bayesian network classifiers: a survey, ACM Comput. Surv. 47 (1) (2014), Article 5.
[4] J.M. Peña, J.A. Lozano, P. Larrañaga, Learning Bayesian networks for clustering by means of constructive induction, Pattern Recognit. Lett. 20 (1999) 1219–1230.
[5] D.T. Pham, G.A. Ruz, Unsupervised training of Bayesian networks for data clustering, in: Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, vol. 465, The Royal Society, 2009, pp. 2927–2948.
[6] D. Heckerman, D. Geiger, D.M. Chickering, Learning Bayesian networks: the combination of knowledge and statistical data, Mach. Learn. 20 (3) (1995) 197–243.
[7] G.F. Cooper, E. Herskovits, A Bayesian method for constructing Bayesian belief networks from databases, in: Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1991, pp. 86–94.
[8] G.F. Cooper, E. Herskovits, A Bayesian method for the induction of probabilistic networks from data, Mach. Learn. 9 (4) (1992) 309–347.
[9] H. Akaike, A new look at the statistical model identification, IEEE Trans. Autom. Control 19 (6) (1974) 716–723.
[10] G. Schwarz, Estimating the dimension of a model, Ann. Stat. 6 (2) (1978) 461–464.
[11] R.R. Bouckaert, Probabilistic network construction using the minimum description length principle, in: Symbolic and Quantitative Approaches to Reasoning and Uncertainty, vol. 747, Springer, 1993, pp. 41–48.
[12] W. Lam, F. Bacchus, Learning Bayesian belief networks: an approach based on the MDL principle, Comput. Intell. 10 (3) (1994) 269–293.
[13] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Algebraic Discrete Methods 8 (2) (1987) 277–284.
[14] K. Grant, M.C. Horsch, Methods for constructing balanced elimination trees and other recursive decompositions, Int. J. Approx. Reason. 50 (9) (2009) 1416–1424.
[15] K. Shoikhet, D. Geiger, A practical algorithm for finding optimal triangulations, in: Proceedings of the 14th National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence, AAAI Press, 1997, pp. 185–190.
[16] F.V. Fomin, Y. Villanger, Treewidth computation and extremal combinatorics, Combinatorica 32 (3) (2012) 289–308.
[17] F.V. Fomin, D. Kratsch, I. Todinca, Exact (exponential) algorithms for treewidth and minimum fill-in, in: Autom. Lang. Program., vol. 3142, Springer, 2004, pp. 568–580.
[18] H.L. Bodlaender, F.V. Fomin, A.M. Koster, D. Kratsch, D.M. Thilikos, On Exact Algorithms for Treewidth, Springer, 2006.
[19] H.M. Markowitz, The elimination form of the inverse and its application to linear programming, Manag. Sci. 3 (3) (1957) 255–269.
[20] U.B. Kjærulff, Triangulation of Graphs-Algorithms Giving Small Total State Space, Tech. rep., 1990.
[21] D.J. Rose, R.E. Tarjan, G.S. Lueker, Algorithmic aspects of vertex elimination on graphs, SIAM J. Comput. 5 (2) (1976) 266–283.
[22] R.E. Tarjan, M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, SIAM J. Comput. 13 (3) (1984) 566–579.
[23] A. Berry, J.R. Blair, P. Heggernes, B.W. Peyton, Maximum cardinality search for computing minimal triangulations of graphs, Algorithmica 39 (4) (2004) 287–298.
[24] U. Kjærulff, Optimal decomposition of probabilistic networks by simulated annealing, Stat. Comput. 2 (1) (1992) 7–17.
[25] P. Larrañaga, C.M. Kuijpers, M. Poza, R.H. Murga, Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms, Stat. Comput. 7 (1) (1997) 19–34.

[26] F. Clautiaux, A. Moukrim, S. Nègre, J. Carlier, Heuristic and metaheuristic methods for computing graph treewidth, RAIRO. Rech. Opér. 38 (1) (2004) 13–26.
[27] A. Koster, Frequency Assignment: Models and Algorithms, Ph.D. thesis, Maastricht University, 1999.
[28] H.L. Bodlaender, A.M. Koster, Treewidth computations I. Upper bounds, Inf. Comput. 208 (3) (2010) 259–275.
[29] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, in: Proceedings of the 25th Annual ACM Symposium on Theory of Computing, ACM, 1993, pp. 226–234.
[30] J. Kwisthout, Most probable explanations in Bayesian networks: complexity and tractability, Int. J. Approx. Reason. 52 (9) (2011) 1452–1469.
[31] G.F. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, Artif. Intell. 42 (2) (1990) 393–405.
[32] P. Dagum, M. Luby, Approximating probabilistic inference in Bayesian belief networks is NP-hard, Artif. Intell. 60 (1) (1993) 141–153.
[33] J. Pearl, Reverend Bayes on inference engines: a distributed hierarchical approach, in: Proceedings of the 2nd AAAI Conference on Artificial Intelligence, AAAI Press, 1982, pp. 133–136.
[34] J.H. Kim, J. Pearl, A computational model for causal and diagnostic reasoning in inference systems, in: Proceedings of the 8th International Joint Conference on Artificial Intelligence, vol. 1, Morgan Kaufmann Publishers Inc., 1983, pp. 190–193.
[35] R.D. Shachter, Evidence absorption and propagation through evidence reversals, in: Proceedings of the 5th Annual Conference on Uncertainty in Artificial Intelligence, North-Holland Publishing Co., 1990, pp. 173–190.
[36] N.L. Zhang, D. Poole, A simple approach to Bayesian network computations, in: Proceedings of the 10th Canadian Conference on Artificial Intelligence, 1994, pp. 171–178.
[37] J. Pearl, A constraint propagation approach to probabilistic reasoning, in: Proceedings of the 1st Annual Conference on Uncertainty in Artificial Intelligence, 1985, pp. 357–369.
[38] A. Darwiche, Recursive conditioning, Artif. Intell. 126 (1) (2001) 5–41.
[39] P.P. Shenoy, G. Shafer, Axioms for probability and belief-function propagation, in: Proceedings of the 4th Annual Conference on Uncertainty in Artificial Intelligence, North-Holland Publishing Co., 1990, pp. 169–198.
[40] F. Jensen, S. Lauritzen, K. Olsen, Bayesian updating in recursive graphical models by local computation, CSQ, Comput. Stat. Q. 4 (1990) 269–282.
[41] C. Boutilier, N. Friedman, M. Goldszmidt, D. Koller, Context-specific independence in Bayesian networks, in: Proceedings of the 12th International Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1996, pp. 115–123.
[42] A. Darwiche, A differential approach to inference in Bayesian networks, J. ACM 50 (3) (2003) 280–305.
[43] M. Niepert, G. van den Broeck, Tractability through exchangeability: a new perspective on efficient probabilistic inference, in: Proceedings of the 28th AAAI Conference on Artificial Intelligence, AAAI Press, 2014, pp. 2467–2475.
[44] J. Korhonen, P. Parviainen, Exact learning of bounded tree-width Bayesian networks, in: Proceedings of the 16th International Conference on Artificial Intelligence and Statistics, vol. 31, Proceedings of Machine Learning Research, 2013, pp. 370–378.
[45] J. Berg, M. Järvisalo, B. Malone, Learning optimal bounded treewidth Bayesian networks via maximum satisfiability, in: Proceedings of the 17th International Conference on Artificial Intelligence and Statistics, vol. 33, 2014, pp. 86–95.
[46] S. Nie, D.D. Mauá, C.P. de Campos, Q. Ji, Advances in learning Bayesian networks of bounded treewidth, in: Advances in Neural Information Processing Systems, 2014, pp. 2285–2293.
[47] P. Parviainen, H. Shahrabi Farahani, J. Lagergren, Learning bounded tree-width Bayesian networks using integer linear programming, in: Proceedings of the 17th International Conference on Artificial Intelligence and Statistics, vol. 33, Proceedings of Machine Learning Research, 2014, pp. 751–759.
[48] G. Elidan, S. Gould, Learning bounded treewidth Bayesian networks, in: Advances in Neural Information Processing Systems, 2009, pp. 417–424.
[49] S. Nie, C.P. de Campos, Q. Ji, Efficient learning of Bayesian networks with bounded tree-width, Int. J. Approx. Reason. 80 (2017) 412–427.
[50] M. Scanagatta, G. Corani, C.P. de Campos, M. Zaffalon, Learning treewidth-bounded Bayesian networks with thousands of variables, in: Proceedings of the 30th International Conference on Neural Information Processing Systems, 2016, pp. 1470–1478.
[51] M. Scanagatta, C.P. d. Campos, G. Corani, M. Zaffalon, Learning Bayesian networks with thousands of variables, in: Proceedings of the 28th International Conference on Neural Information Processing Systems, MIT Press, 2015, pp. 1864–1872.
[52] J. Cussens, Bayesian network learning with cutting planes, in: Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2011, pp. 153–160.
[53] M. Scanagatta, G. Corani, M. Zaffalon, J. Yoo, U. Kang, Efficient learning of bounded-treewidth Bayesian networks from complete and incomplete data sets, Int. J. Approx. Reason. 95 (2018) 152–166.
[54] F.R. Bach, M.I. Jordan, Thin junction trees, in: Advances in Neural Information Processing Systems, 2001, pp. 569–576.
[55] D. Karger, N. Srebro, Learning Markov networks: maximum bounded tree-width graphs, in: Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2001, pp. 392–401.
[56] A. Chechetka, C. Guestrin, Efficient principled learning of thin junction trees, in: Advances in Neural Information Processing Systems, 2008, pp. 273–280.
[57] D. Shahaf, C. Guestrin, Learning thin junction trees via graph cuts, in: Proceedings of the 12th International Conference on Artificial Intelligence and Statistics, vol. 5, Proceedings of Machine Learning Research, 2009, pp. 113–120.
[58] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, 1993.
[59] D. Lowd, P. Domingos, Learning arithmetic circuits, in: Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2008, pp. 383–392.
[60] M. Benjumeda, P. Larrañaga, C. Bielza, Learning Bayesian networks with low inference complexity, Progr. Artif. Intell. 5 (1) (2015) 15–26.
[61] A. Darwiche, Modeling and Reasoning with Bayesian Networks, Cambridge University Press, 2009.
[62] D. Lowd, J. Davis, Learning Markov network structure with decision trees, in: IEEE International Conference on Data Mining, IEEE, 2010, pp. 334–343.
[63] J. Van Haaren, J. Davis, Markov network structure learning: a randomized feature generation approach, in: Proceedings of the 26th AAAI Conference on Artificial Intelligence, 2012, pp. 1148–1154.
[64] J. Bekker, J. Davis, A. Choi, A. Darwiche, G.V.d. Broeck, Tractable learning for complex probability queries, in: Proceedings of the 28th International Conference on Neural Information Processing Systems, MIT Press, 2015, pp. 2242–2250.
[65] S. Holm, A simple sequentially rejective multiple test procedure, Scand. J. Stat. (1979) 65–70.
[66] J.P. Shaffer, Modified sequentially rejective multiple test procedures, J. Am. Stat. Assoc. 81 (395) (1986) 826–831.
[67] S. Garcia, F. Herrera, An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons, J. Mach. Learn. Res. 9 (2008) 2677–2694.
[68] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[69] L.C. van der Gaag, P.R. de Waal, Multi-dimensional Bayesian network classifiers, in: Proceedings of the 3rd European Workshop on Probabilistic Graphical Models, 2006, pp. 107–114.
[70] C. Bielza, G. Li, P. Larrañaga, Multi-dimensional classification with Bayesian networks, Int. J. Approx. Reason. 52 (6) (2011) 705–727.
[71] M. Benjumeda, C. Bielza, P. Larrañaga, Tractability of most probable explanations in multidimensional Bayesian network classifiers, Int. J. Approx. Reason. 93 (Supplement C) (2018) 74–87.
[72] N. Friedman, The Bayesian structural EM algorithm, in: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1998, pp. 129–138.