Learning Low Inference Complexity Bayesian Networks

Marco Benjumeda, Pedro Larrañaga, and Concha Bielza

Departamento de Inteligencia Artificial,
Universidad Politécnica de Madrid,
Campus de Montegancedo, 28660 Boadilla del Monte, Madrid, Spain
marco.benjumeda.barquita@upm.es,{pedro.larranaga,mcbielza}@fi.upm.es
http://cig.fi.upm.es/

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

Keywords: Probabilistic graphical models, Bayesian networks, arithmetic circuits, network polynomials, structure learning, thin models

1 Introduction

Bayesian networks (BNs) are very powerful tools for concisely modelling probability distributions of a set of random variables. In the past years there has been a huge interest in the creation of new methods for learning the structure of BNs from data. The main approaches focus on improving the accuracy (mainly the likelihood) of the networks, learning sometimes spurius relations among variables and increasing the inference complexity of the resulting models. Some methods include a penalization for the representation complexity of the network using the number of parameters of the model [1, 6], but the representation complexity and the inference complexity are sometimes very different for the same model [5]. In practice, performing exact inference in the models learned using this type of methods is usually computationally expensive and sometimes intractable. The most common solution is then to resort to approximate inference, deteriorating the inference accuracy of the models.

A possible solution to learn models with tractable exact inference is to use an estimation of the inference complexity as a penalization during the learning

process. In junction trees (JTs) the inference complexity is exponential in the size of the largest clique of the tree, which is called treewidth, and it is a good indicator of the inference complexity for probabilistic models. There are some approaches that learn low-treewidth JTs, which are usually called thin JTs [3], using a bounded treewidth to restrict the inference complexity [7, 9, 15]. A different approach, closely related to our work, uses the incremental compilation of arithmetic circuits (ACs) to obtain tractable models [13].

In this work we propose a new representation to complement Bayesian networks during the learning process, such that it can be used as an inference complexity indicator, and also providing a framework for exact inference.

The rest of the paper is organized as follows. Section 2 is an introduction to network polynomials (NPs). Section 3 describes the new model, which we call polynomial trees (PTs), and introduces learning and inference procedures for PTs. Section 4 shows the experimental results. Section 5 gives the conclusive remarks and future research lines.

2 Network Polynomials

The probability distribution implicit in any BN $\mathcal{B}$ can be also represented as a network polynomial [8], that is, a multi-linear function over two types of variables, indicators I_{x_i} and parameters $\theta_{x_i|\pi_i} = P(X_i = x_i|\mathrm{Pa}_{\mathcal{B}}(X_i) = \pi_i)$, where $\mathrm{Pa}_{\mathcal{B}}(X_i)$ are the parents of X_i in $\mathcal{B}$. The indicators are Boolean functions that return 1 if $X_i = x_i$ or if the value of X_i is unknown, and 0 otherwise. The probability distribution of discrete variables $X_1, \ldots, X_n$ in an NP is described as:

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

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

This function represents the joint probability over a set of variables. NPs allow answering any arbitrary marginal or conditional probabilistic query in linear time in the size (number of sums and products) of the polynomial.

2.1 Arithmetic Circuits

The size of the NPs grows exponentially with the number of variables, making inference nearly intractable for common-size networks. Trying to overcome this difficulty, Darwiche [8] proposed the use of ACs to represent NPs, using the distributive properties of the polynomials to reduce the complexity of the NP function.

ACs are directed acyclic graphs (DAGs) in which the inner nodes (nodes with children) are addition ($+$) and multiplication ($\times$) nodes and the leaves (nodes

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

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

$$P(A,B) = I_a I_b \theta_a \theta_{b|a} + I_a I_{\neg b} \theta_a \theta_{\neg b|a} + I_{\neg a} I_b \theta_{\neg a} \theta_{b|\neg a} + I_{\neg a} I_{\neg b} \theta_{\neg a} \theta_{\neg b|\neg a} \ . \quad (1)$$

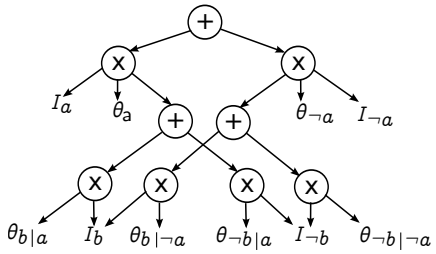


Fig. 1: AC representation for the NP of (1).

Darwiche [8] uses ACs as a complementary representation obtained by the compilation of BNs to perform tractable exact inference. The first approach to learn ACs directly from data was proposed by Lowd and Domingos [13]. It uses a greedy search process that penalizes each circuit with its inference complexity, given by the number of arcs of the circuit. A drawback of using ACs is that their size can be huge when the number of variables in the model is not small.

3 Polynomial Trees

State-of-the-art methods for learning thin models, such as ACs and thin JTs, consider a very restricted search space in each iteration due to the difficulty of making incremental changes in these models. PTs have a simpler representation, so the cost of incrementally compiling any local change done to a BN (arc additions, removals or reversions) is lower, allowing a more flexible learning process of low inference complexity models.

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

1. A set of nodes $\mathcal{X}_P = \{*\} \cup \mathcal{X}$, where $*$ is the root node.

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

Fig. 2 is an example of a BN (a) and a PT associated to the BN (b).

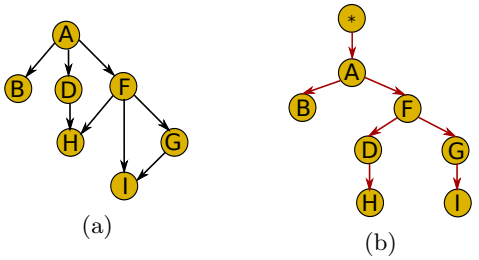


Fig. 2: Structure of a BN (a), and a PT associated to the BN (b)

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

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

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

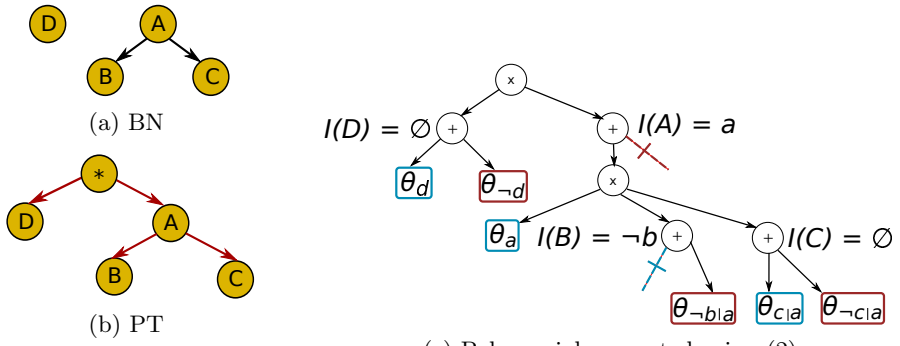
3.1 Inference in Polynomial Trees

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

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

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

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



(a) BN (b) PT (c) Polynomial generated using (2)

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

3.2 Evaluating the Complexity of Polynomial Trees

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

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

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

3.3 Incremental Compilation of PTs

To evaluate the inference complexity of each network using (3) it is necessary to have a compiled PT in each step of the learning process. As compiling a PT from scratch every time is intractable, we have created a group of procedures to compile incrementally in PTs any local change (arc additions, deletions and reversals) that could be done in a BN during the learning process. Next, we show the general procedure used to compile PTs incrementally. The methods are described in detail in [4].

Arc Addition. When compiling in a PT $\mathcal{P}$ the addition of an arc $(X_{out} \rightarrow X_{in})$ in a BN $\mathcal{B}$ we can face three possible scenarios:

1. $X_{out} \in \text{Pred}_{\mathcal{P}}(X_{in})$: $\mathcal{P}$ is sound for $\mathcal{B}$ after the addition, so no changes are required.
2. $X_{in} \in \text{Pred}_{\mathcal{P}}(X_{out})$: In this scenario, it is necessary to set X_{out} as a predecessor of X_{in} in $\mathcal{P}$ and reconfigure the positions of the nodes between X_{out} and X_{in} to obtain a PT sound for $\mathcal{B}$. First, let $\mathcal{S}$ be the set of nodes that are predecessors of X_{out} and descendants of X_{in} in $\mathcal{P}$. For each node $X_i \in \mathcal{S}$, if X_i is a descendant of X_{in} in $\mathcal{B}$ it must be also set as a descendant of X_{in} in $\mathcal{P}$. Otherwise, for each children X_j of X_i in $\mathcal{P}$, we set $\text{Pa}_{\mathcal{P}}(X_j)$ to the deepest predecessor of X_{in} in $\mathcal{P}$ that is also a descendant of X_{in} in $\mathcal{B}$.
3. $X_{out} \notin \text{Pred}(X_{in})$ and $X_{in} \notin \text{Pred}(X_{out})$: In this case node X_{out} and its predecessors in $\mathcal{P}$ are set as predecessors of X_{in} in $\mathcal{P}$.

Arc Deletion. After an arc deletion in a BN $\mathcal{B}$, it is not necessary to make any changes in its corresponding PT $\mathcal{P}$.

Arc Reversal. To compile the reversal of arc $X_{out} \rightarrow X_{in}$ we compile first the deletion of arc $X_{out} \rightarrow X_{in}$ and then the addition of the reversed arc $X_{in} \rightarrow X_{out}$.

3.4 Polynomial Tree Optimization

Although the methods proposed above assure the soundness of the compiled PTs, the obtained models may be far from optimal. We say that a PT $\mathcal{P}$ is optimal for a BN $\mathcal{B}$ if it is the PT with minimum inference complexity (measured by (3)) that is sound for $\mathcal{B}$.

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

The key of the optimization process is to find the right local movements that minimize (3) in each iteration. The main idea of the procedure is to swap the position in $\mathcal{P}$ of the node to be optimized X_{opt} with its parent $\text{Pa}_{\mathcal{P}}(X_{opt})$ and check if the change reduces (3).

3.5 Learning Polynomial Trees from Data

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

Therefore, we penalize the log-likelihood (LL) to measure the accuracy of each model, favoring candidates with low inference complexity. For a dataset D of size N, the scoring function is defined as:

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

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

To learn the structure of PTs, we use the methods proposed above in combination with the 2 iterations constrained hill-climbing (2iCHC) algorithm [11]. 2iCHC learns the structure of BNs using the hill-climbing (HC) algorithm with a forbidden parents list to constrain the search space, reducing the learning time of HC while assuring the return of a minimal I-map. We call the resulting method hill-climbing for polynomial trees (HCPT).

4 Experimental Results

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

The datasets used in this work were generated from three real-world BNs. WIN95PTS is a medium network for handling printer troubleshooting in Windows 95, PATHFINDER is a large network for the diagnosis of lymph-node diseases [12], and MUNIN1 is a large size network for the diagnosis of neuromuscular disorders [2]. The basic properties of each BN are shown in Table 1. We have generated 25000 learning samples and 40000 testing samples from each network.

Table 1: Basic properties of the BNs used for the experiments

	WIN95PTS	PATHFINDER	MUNIN1
Number of nodes	76	135	186
Number of arcs	112	200	273
Number of parameters	574	77155	15622
Average Markov blanket size	5.92	3.04	3.81

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

from 10% to 25% of the total, letting the number of query variables fixed at 15%. The MSE is defined by:

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

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

4.1 Learning Results

This work focuses on providing a framework for the incremental compilation of PTs that can be easily applied in most score+search methods. Therefore, we were interested in comparing an existing BN learning method to a modified version of this method using PTs. The models obtained by the incremental compilation of PTs are learned using the HCPT method, that is an adaptation of 2iCHC, and they are compared with the BNs learned using the 2iCHC algorithm in combination with the minimum description length (MDL) [6]. Parameters k_n and k_p were set empirically to 0.5 and 1 respectively.

Table 2: Learning results

	WIN95PTS		PATHFINDER		MUNIN1	
	2iCHC	HCPT	2iCHC	HCPT	2iCHC	HCPT
Log_Likelihood	−9.11	−9.62	−27.19	−26.75	−41.78	−45.73
Number of arcs	120	131	138	140	220	210
Number of parameters	620	435	1266	1273	2085	2190
Learning time	0 h 12 m	0 h 14 m	1 h 51 m	2 h 34 m	5 h 37 m	6 h 53 m

The results (Table 2) show that the differences in the likelihood are small. The time required for the learning process is a bit higher in HCPT than in 2iCHC, but the time needed for the incremental compilation of PTs is small compared with the time spent by the scores. Nevertheless, we focus on reducing the inference complexity rather than the learning time, given that the learning process is usually performed only once, while inference is usually performed multiple times.

4.2 Inference Results

Next, we compare the performance of exact inference in the learned PTs against approximate inference in the BNs learned with the 2iCHC algorithm. We do not use exact inference in the models learned with 2iCHC because its computational cost is too high for large networks. We use the likelihood weighting (LW) algorithm [10, 14] for approximate inference. The results are presented in Fig. 4.

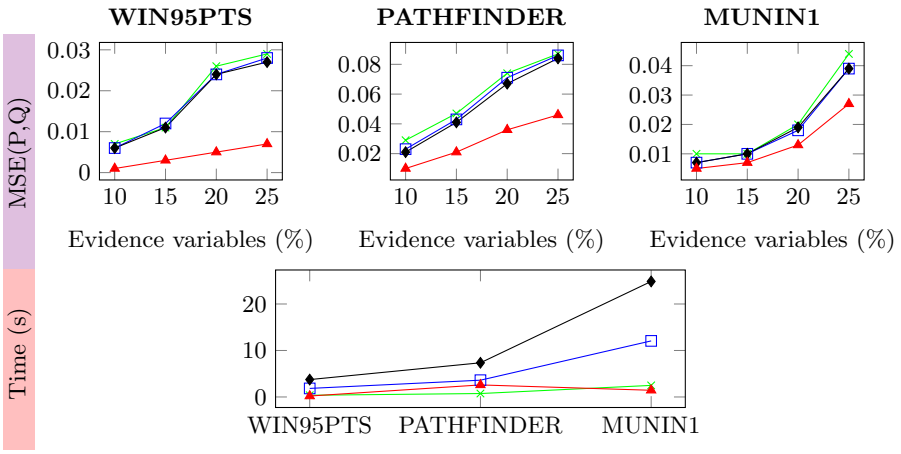


Fig. 4: Inference results comparing quick LW (100 samples) —×—, medium LW (1000 samples) —□—, slow LW (2000 samples) —♦—, and exact inference with PTs —▲—. The computational cost displayed is the mean time (in seconds) of all the queries answered by each BN.

The inference results show that using PTs improves the accuracy of the answers provided by the models obtained with 2iCHC in combination with LW. The computational cost of performing exact inference in PTs is lower than the cost of using 2iCHC and medium (1000 samples) or slow (2000 samples) LW, and similar to the cost of performing quick LW (200 samples), that produces always the less accurate answers in the tests.

5 Conclusions and Future Research

We developed a new model for the graphical representation of NPs. The new representation is simple and intuitive, and it allows evaluating the inference complexity of the candidate models during the learning process, providing also an exact inference framework. The experimental results show that using the incremental compilation of PTs combined with existing BN learning methods obtains models with low inference complexity. In the tests, the accuracy of the answers provided by exact inference in PTs outperformed those provided by the models learned with a state-of-the-art BN learning method using approximate inference.

Future research may focus on learning PTs for multidimensional classification and learning PTs with latent variables.

Acknowledgments. This work has been partially supported by the Spanish Ministry of Economy and Competitiveness through the Cajal Blue Brain (C080020-09; the Spanish partner of the Blue Brain initiative from EPFL) and

TIN2013-41592-P projects, by the Regional Government of Madrid through the S2013/ICE-2845-CASI-CAM-CM project, and by the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 604102 (Human Brain Project). M. Benjumeda is supported by a predoctoral contract for the formation of doctors (call 2014) from the Spanish Ministry of Economy and Competitiveness (BES-2014-068637).

References

1. Akaike, H.: A new look at the statistical model identification. IEEE Transactions on Automatic Control 19(6), 716–723 (1974)
2. Andreassen, S., Rosenfalck, A., Falck, B., Olesen, K. G., Andersen, S. K.: Evaluation of the diagnostic performance of the expert EMG assistant MUNIN. Electroencephalography and Clinical Neurophysiology/Electromyography and Motor Control, 101(2), 129–144 (1996)
3. Bach, F.R., Jordan, M.I.: Thin junction trees. In: Advances in Neural Information Processing Systems. pp. 569–576 (2001)
4. Benjumeda, M.: Learning Bayesian Networks From Data by the Incremental Compilation of Polynomial Trees. Master's thesis, Universidad Politécnica de Madrid (2014)
5. Beygelzimer, A., Rish, I.: Approximability of probability distributions. In: Advances in Neural Information Processing Systems. pp. 377–384 (2004)
6. Bouckaert, R.R.: Probabilistic network construction using the minimum description length principle. In: Symbolic and Quantitative Approaches to Reasoning and Uncertainty, pp. 41–48. Springer (1993)
7. Chechetka, A., Guestrin, C.: Efficient principled learning of thin junction trees. In: Advances in Neural Information Processing Systems. pp. 273–280 (2008)
8. Darwiche, A.: A differential approach to inference in Bayesian networks. Journal of the Association for Computing Machinery 50(3), 280–305 (2003)
9. Elidan, G., Gould, S.: Learning bounded treewidth Bayesian networks. In: Advances in Neural Information Processing Systems. pp. 417–424 (2009)
10. Fung, R.M., Chang, K.C.: Weighing and integrating evidence for stochastic simulation in Bayesian networks. In: Uncertainty in Artificial Intelligence. pp. 209–220 (1989)
11. Gámez, J.A., Mateo, J.L., Puerta, J.M.: Learning Bayesian networks by hill climbing: Efficient methods based on progressive restriction of the neighborhood. Data Mining and Knowledge Discovery 22(1-2), 106–148 (2011)
12. Heckerman, D., Horwitz, E., Nathwani, B.: Towards normative expert systems: Part I. The pathfinder project. Methods of Information in Medicine 31, 90–105 (1992)
13. Lowd, D., Domingos, P.: Learning arithmetic circuits. In: Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence. pp. 383–392 (2008)
14. Shachter, R.D., Peot, M.A.: Simulation approaches to general probabilistic inference on belief networks. In: Uncertainty in Artificial Intelligence. pp. 221–234 (1989)
15. Shahaf, D., Guestrin, C.: Learning thin junction trees via graph cuts. In: International Conference on Artificial Intelligence and Statistics. pp. 113–120 (2009)