Learning more Accurate Bayesian Networks in the CHC Approach by Adjusting the Trade-Off between Efficiency and Accuracy

Jacinto Arias, José A. Gámez, and José M. Puerta

Department of Computing Systems - I3A
University of Castilla-La Mancha
Albacete, 02071
jacinto.arias@alu.uclm.es, {jose.gamez,jose.puerta}@uclm.es

Abstract. Learning Bayesian networks is known to be an NP-hard problem, this, combined with the growing interest in learning models from high-dimensional domains, leads to the necessity of finding more efficient learning algorithms. Recent papers propose constrained approaches of successfully and widely used local search algorithms, such as hill climbing. One of these algorithms families, called CHC (Constrained Hill Climbing), highly improves the efficiency of the original approach, obtaining models with slightly lower quality but maintaining its theoretical properties. In this paper we propose some modifications to the last version of these algorithms, FastCHC, trying to improve the quality of its output by relaxing the constraints imposed to include some diversification in the search process. We also perform an intensive experimental evaluation of the modifications proposed including quite large datasets.

Keywords: Bayesian Networks, Machine Learning, Local Search, Constrained Search, Scalability.

1 Introduction

Over the last decades, Bayesian Networks [7,9] have become one of the most relevant knowledge representation formalisms in the field of Data Mining. Due to its popularity and the increasing amount of data available it is not surprising that learning the structure of Bayesian Networks from data has become also a problem of growing interest.

This paper falls in the so-called *score+search* approach which poses learning as an optimization problem: A scoring metric function f ([6,8]) is used to score a network structure with respect to the training data, and a search method is used to look for the network with the best score. The majority of the proposed methods are based on heuristic and metaheuristic search strategies since this problem is known to be NP-hard [2]. If we add the necessity of dealing with massive datasets, local-search methods such as hill climbing, have achieved great popularity due to its ease of implementation and its trade-off between their efficiency and the quality of the models obtained.

In order to deal with larger datasets, several scalable algorithms have been proposed, specifically based on local-search approaches. Those methods are usually based on restricting the search space in different ways performing a two-step process to first detect the constraints, and then perform an intensified and more efficient local search [10]. In [3], the CHC algorithm is introduced, which progressively restricts the search space when performing an iterated local search without needing a previous step. The development of this result leads to the FastCHC algorithm [4] which improves the performance of the original definition by reducing to one the number of iterations needed.

All the CHC algorithms provide an efficiency improvement when comparing them with most state of the art algorithms and especially with the unconstrained hill climbing approach, maintaining also its original theoretical properties. However, restricting the search space normally implies a loss of quality in the models obtained. In this paper, we propose some modifications to the FastCHC algorithm, as it has proven to be the most efficient of the CHC family, in order to balance the trade-off between efficiency and quality of the models, trying to provide better solutions without decreasing its efficiency advantage. These modifications relax the constraints imposed by the original algorithm in order to allow the algorithm visit additional solutions.

The rest of the paper is organized as follows: In Section 2 we review the necessary background on Bayesian Networks and the hill climbing approach to structural learning. In Section 3 we review the CHC algorithms family. In Section 4 we describe the modifications proposed to improve the FastCHC algorithm. Finally, in section 5 we evaluate the performance of the modifications proposed and compare them with the original algorithms. In section 6, we conclude with a discussion of the results obtained and future directions.

2 Learning the Structure of Bayesian Networks

Bayesian Networks (BNs) are graphical models than can efficiently represent and manipulate n-dimensional probability distributions [9]. Formally[1], a BN is a pair $B = \langle \mathcal{G}, \mathbf{\Theta} \rangle$, where $\mathcal{G}$ is a graphical structure, or more precisely a Directed Acyclic Graph (DAG) whose nodes are in $\mathbf{V} = \{X_1, X_2, \ldots, X_n\}$ represent the random variables of the domain we wish to model, and the topology of the graph (the arcs in $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$) encodes conditional (in)dependence relationships among the variables (by means of the presence or absence of direct connections between pair of variables).

The second element of the pair, $\mathbf{\Theta}$, represents a set of numerical parameters, usually conditional probability distributions drawn from the graph structure

[1] We use standard notation, that is, bold font to denote sets and n-dimensional configurations, calligraphic font to denote mathematical structures, upper case for variables sets of random variables, and lower case to denote states of variables or configuration of states.

which quantifies the network: For each $X_i \in \mathbf{V}$ we have a conditional probability distribution $P(X_i \mid pa(X_i))$, where $pa(X_i)$ represents any combination of the values of the variables $Pa(X_i)$, and $Pa(X_i)$ is the parent set of X_i in $\mathcal{G}$.

From a BN $B = \langle \mathcal{G}, \mathbf{\Theta} \rangle$, we can recover the joint probability distribution over $\mathbf{V}$ given by:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Pa(X_i))$$

The problem of learning the structure of a BN can be stated as follows: Given a training dataset $D = \{v^1, \ldots, v^m\}$ of instances of $\mathbf{V}$, find the DAG $\mathcal{G}^*$ such that

$$\mathcal{G}^* = \arg\max_{\mathcal{G} \in \mathcal{G}^n} f(\mathcal{G} : D)$$

where $f(\mathcal{G} : D)$ is a scoring metric which evaluates the merit of any candidate DAG. An important property of the metrics that are commonly used for BN structural learning is the decomposability in presence of full data, which evaluate a given DAG as the sum of its node family scores, i.e. the subgraphs formed by a node and its parents in $\mathcal{G}$ [7]. This provides an efficient neighbourhood evaluation for local search algorithms such as hill climbing, which evaluates local changes for a candidate solution, normally starting from the empty graph, and performs the one which maximizes the score function until it is not possible to find a better neighbour. By using local changes which only modify one arc at each step, we can reuse the computations carried out in previous stages and compute only the statistics corresponding to the variables whose parents have been modified. The most used operators are arc addition, deletion and reversal.

The popularity of HC is probably due to its ease of implementation as well as its good trade-off between efficiency and quality of the output, which is a local optimum. In addition, it has other theoretical properties which makes it interesting, e.g. under certain assumptions the algorithm guarantees that the resulting network is a minimal I-map[2] [3].

3 Constrained Hill Climbing Methods

The CHC algorithm [3] is based on a progressive restriction of the neighbourhood during the search process. The algorithm keeps what the authors call, *Forbidden Parents Sets (FP)* for each node, so in the neighbourhood generation step for the node X_i, any node $X_j \in FP(X_i)$ is not considered as a suitable parent of X_i and the algorithm avoids its evaluation, saving a large number of computations.

In order to include a node in the FP set, the algorithm uses the value of the score metric as a sort of conditional independence test when evaluating

[2] A DAG $\mathcal{G}$ is an I-map of a probability distribution p if all independences that $\mathcal{G}$ codifies are also present in the original p distribution and it is minimal if none of the arcs in the DAG can be deleted without violating the I-map condition.

local changes, so that when the difference ($diff$) of score between the current structure and the one resulting of applying the considered operation does not reveal a gain in the structure, the FP sets are updated consequently:

- Adding $X_j \rightarrow X_i$. If $diff < 0$ then $\{X_j\}$ is added to $FP(X_i)$ and vice versa.
- Deleting $X_j \rightarrow X_i$. If $diff > 0$ then $\{X_j\}$ is added to $FP(X_i)$ and vice versa.
- Reversal of $X_j \rightarrow X_i$. Decompose as deleting($X_j \rightarrow X_i$)+adding($X_i \rightarrow X_j$) and use the previous two rules to update the FP sets.

Although the initial CHC algorithm results in a much more efficient search process when compared with the unconstrained hill climbing it does not guarantee to return a minimal I-map and for that reason the CHC* algorithm is proposed, in which the output of the CHC algorithm is used as the initial solution of an unconstrained hill climbing to retain the theoretical properties.

An iterated version of the CHC algorithm is also proposed in [3], in which the algorithm performs several iterations by using the output of the previous one as the initial solution for the next restarting the FP sets to the empty set. The algorithm ends when it is unable to perform any change at the beginning of an iteration and, because no constraints are being used, it has the same stopping criterion as the hill climbing algorithm thus retains its theoretical properties: It guarantees a minimal I-map.

All the previous development lead to the most efficient version of the CHC algorithm, called FastCHC [4]. This algorithm guarantees a minimal I-map in just one iteration. To accomplish this, the algorithm tries to correct wrong discovered relationships in the graph by releasing some constraints every time it performs an addition operation to the network, i.e. after the algorithm adds the arc $X_i \rightarrow X_j$ it releases all the constraints from $FP(X_i)$ and $FP(X_j)$ respective neighbourhoods to allow the algorithm to correct the solution if needed to become an I-map.

4 Proposal

As mentioned before, constrained algorithms experiment a loss of quality for the sake of efficiency [4]. In this paper we propose two different strategies to relax the constraints present in the FastCHC algorithm to allow some diversification in the search process, thus it will visit additional solutions that otherwise would remain unexplored. All the following modifications maintain the theoretical properties of the original algorithm as we don't modify the required conditions.

4.1 Releasing Constraints in Variable Neighbourhood Levels

Our first proposal is a basic modification to FastCHC based on the aforementioned strategy [4] of releasing some of the constraints from the nodes involved in the change performed at each search step. Although this strategy was originally designed to correct wrong discovered relationships, we can also see it as a *diversification* technique, as it allows the search algorithm to explore additional solutions which could have a better score.

This modification, which we call FastCHC$_{Mod1}$, extends the described procedure to deletion and reversal operations. We also consider releasing constraints from a wider range of nodes in order to extend the unconstrained search space. To achieve this, we include a new parameter $L \in \mathbb{N}$ which represents levels of neighbourhood to release constraints from, i.e., for a value $L = 1$ when a change involving nodes X_i and X_j is performed any node adjacent to X_j ($adj^1(X_j)$) will be removed from $FP(X_i)$, and vice versa[3]; for a level $L = 2$, any node $X_{j'} \in adj^1(X_j)$ and its respective adjacent nodes ($adj^2(X_j)$) will be removed from $FP(X_i)$, and vice versa; for the general case $L = n$ any node in $adj^n(X_j)$ will be removed from $FP(X_i)$, and vice versa.

4.2 Limiting the FP Sets Size

The main disadvantage of the previous approach is that the behaviour of the modification is not much predictable and for that reason the new included parameter L could be difficult to set. Our second proposal tries to release constraints during the search procedure regarding the score metric value.

We add a limit S for the maximum number of FP constraints that can be simultaneously stored in the FP, so when the number of constraints reaches this limit some of them must be released in order to keep the number of constraints at S. As the constraints are added or released in pairs, i.e. if variable X_i is added to $FP(X_j)$ also X_j will be added to $FP(X_i)$, we count both directions as one so the parameter S refers to half the size of the sum of all the FP sizes:

$$\frac{1}{2}\sum_{i=1}^{n} \#FP(X_i) \leq S$$

This modification requires two design decisions to be defined: A suitable approximation S for the maximum number of FP constraints to be maintained and an update criteria to determine which constraints must be kept in the FP sets.

We should not use an absolute approximation to select an appropriate S value because the number of constraints that are discovered during a search procedure highly depends on the dataset number of attributes and other specific characteristics. For that reason, we ought to express a limitation parameter independently from the dataset that is being used. To make an approximate idea of the size that the FP sets reach when using different datasets, we carried out an experiment computing the maximum number of discovered constrains (S_{max}) for different datasets, which are described in Section 5. In Table 1 we can confirm how much this number varies from one dataset to another.

We can use this value S_{max} as a reference value that we can compute automatically and then express the size limitation as a reduction factor α which will

[3] The behaviour of the modification with $L = 1$ is similar to the original FastCHC, which releases constraints in the same way but only after performing addition moves.

Table 1. FP sets maximum size reached during a full execution of the original FastCHC and after the first iteration of the algorithm. The last column includes the ratio between these two values as a comparison.

Network	#vars	Full Execution (S_{max})	First Iteration (S_0)	Ratio
Mildew	35	556	474	85%
Barley	48	1053	841	80%
Hailfinder	56	1458	1243	85%
Pigs	441	96142	86912	90%

be applied to obtain S; having a parameter independent from the dataset. However, since obtaining S_{max} is not feasible without performing a full execution of the algorithm, we must find an approximate value that we could obtain using computations that the unmodified algorithm already performs. In Table 1, we show the size of the FP sets after the first iteration of the algorithm, S_0, and the ratio between this value and the one obtained after a full execution S_{max}, showing that there is no much difference between them, thus we can take the latter value as an optimistic approximation that should fit our requirements.

In summary, our modification performs the first iteration of the search process just as it does FastCHC, discovering and storing in the FP sets S_0 constraints, then a reduction of α is applied to the size of the FP sets and the algorithm releases all the constrains needed to fit this new maximum size: $S = \alpha \cdot S_0$. From that point, FastCHC algorithm is executed, with the difference that, when new FP constrains are discovered, the FP sets must be updated and some of the constraints need to be released in order to fit the imposed limit.

Regarding the update policy, we keep a list including all the constraints discovered ordered by their score at time of being included (as they are forbidden they will not be updated anymore). When the amount of constraints exceeds the limit, the one with the highest score will be released from the list.

In order to manually fix the parameter α we can interpret it as a balance between efficiency and quality of the models. As we can see in Figure 1, a value of α closer to 1.0 must keep an algorithm behaving much like FastCHC, but a value of α closer to 0.0 keeps the algorithm's behaviour closer to the Hill Climbing approach but retaining the speed up advantage of the first iteration of the original constrained algorithm.

5 Experimental Evaluation

In this section we examine the different modifications to the FastCHC algorithm proposed in this paper, we perform an empirical evaluation for each modification with different values of their parameters. We selected FastCHC$_{Mod1}$ with $L = 2$ and $L = 3$, as preliminary experiments revealed that larger values of L don't provide much difference, and FastCHC$_{Mod2}$ with $\alpha = 0.4$, $\alpha = 0.6$, and $\alpha = 0.8$ to evaluate α in a wide range. In addition, we include as reference algorithms the unmodified FastCHC and the standard hill climbing (HC).

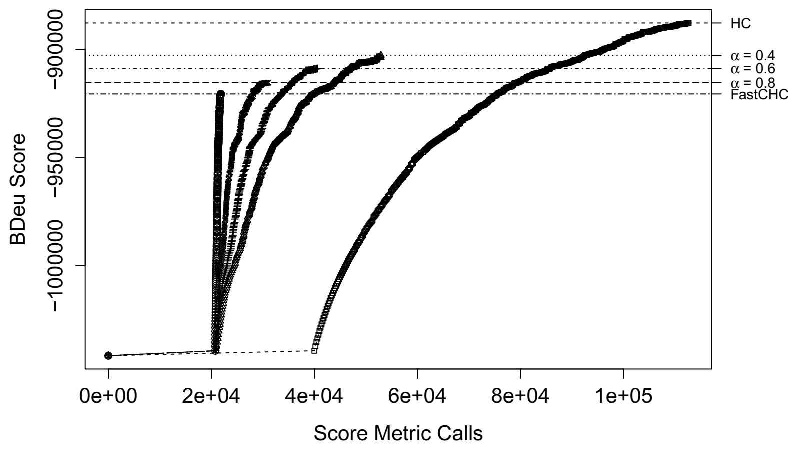

Fig. 1. Evolution of the search process of the HC, FastCHC and FastCHC$_{Mod2}$ with different values of α. Each point correspond to a complete search step representing the BDeu score metric value obtained (y-axis) and the score metric calls (x-axis). This execution corresponds to an single execution for a 5000 instances sample from one of the random *BN_200* networks, described in Section 5.

5.1 Implementation and Running Environment

All the algorithms have been implemented in Java using the ProGraMo library for dataset and graph structures management [5]. The score metric used is the Bayesian Dirichlet Equivalent Uniform (BDeu) [6] with an equivalent sample size of 10 and all other parameters set up as in [10]. We also take advantage of the internal cache described in [3] which saves the result of every score computation using the probability family as a hash key in order to re-use it later in the execution, achieving high computational savings especially in larger domains.

5.2 Performance Indicators

We consider two kind of factors as performance indicators to compare the different algorithms: the quality of the network obtained which is given by the value of the scoring metric (BDeu) and the efficiency of the algorithm which is given by the number of score function computations carried out by each algorithm (calls). As the execution time depends on both the implementation and the specification of the computer on which the algorithm is executed, we consider the score function calls for being independent of those factors and having direct correspondence with CPU time requirements.

5.3 Experiments

We have carried out a first set of experiments using a collection of *real worl networks* which are commonly referenced in the literature and varies from smaller

Table 2. Main characteristics of the real networks used in the experiments

Network	Alarm	Barley	Hailfinder	Insurance	Mildew	Munin1	Pigs
#vars	38	48	56	27	35	189	441
#arcs	46	84	66	52	46	282	592
Domain	Medicine	Agriculture	Meteorology	Insurance	Agriculture	Medicine	Genetics

to larger domains, the networks have been obtained from the Bayesian Network Repository[4]. Their main characteristics are shown in Table 2. In addition, a second set of more intensive experiments have been carried on using a set of synthetic databases sampled from a collection of *artificial networks*, which have been randomly generated with different degrees of difficulty based on the procedure described in [1]. We use a collection of 4 networks with 100 nodes and an average of 100 arcs and 4 networks with 200 nodes and an average of 400 arcs. We have sampled 5 datasets of 5000 instances for each network, the following results are obtained from the average of them.

5.4 Detailed Results

Table 3 shows the BDeu score metric value and calls for each algorithm and database. The results highlighted in bold are the best for the corresponding network. As we can confirm in the data, the algorithms perform consistently with their definition being hill climbing the one with highest scores and FastCHC the most efficient. Also, we can confirm the score improvements of the modifications, especially when comparing FastCHC$_{Mod2}$ for the three different values of α; FastCHC$_{Mod1}$ score improvement is more subtle but efficiency is hardly modified. As we can see, the score value an calls difference between the constrained and the unconstrained algorithms is more noticeable for the larger databases, supporting the scalability properties of the constrained algorithms.

5.5 Summary

Taking into account the two sets of experiments described above, a comprehensive performance comparison for each algorithm with hill climbing is shown in Figure 2, regarding the ratio between the score metric calls and the BDeu score; the later is computed using $\exp((\texttt{BDeu}(\texttt{Model}_i) - \texttt{BDeu}(\texttt{HC}))/m)$ in order to provide an estimation of the ratio between the probability that $\texttt{Model}_i$ and model HC assign to the next data sample. In this graphical comparison we can confirm the expected behaviour of the modifications, displaying the FastCHC algorithm in the bottom left corner as the most efficient but less accurate algorithm and the hill climbing in the upper right corner, being the less efficient algorithm which obtains the best solutions; the modifications are spread along the diagonal showing the desired balance between efficiency and quality according to their parameters values meaning.

[4] `http://www.cs.huji.ac.il/site/labs/compbio/Repository/`

Table 3. Score metric value (above) and calls (below) for each algorithm and network. BN_100 and BN_200 represent the average of the results obtained for the two collection of synthetic networks described.

Dataset	HC	FastCHC	FastCHC$_{Mod1}$ ($L = 2$)	FastCHC$_{Mod1}$ ($L = 3$)
Alarm	**-47999.3454**	-48045.1449	-48010.4824	-48009.5839
Barley	**-261888.6082**	-271658.9229	-269808.1994	-267565.7600
Hailfinder	**-250408.8787**	-251528.0754	-251417.5400	-251411.9393
Insurance	-67021.9905	-67317.2964	-67172.8644	-67131.2555
Mildew	**-232748.2202**	-243978.3822	-243444.3285	-243278.5567
Munin1	**-203005.5923**	-206639.3665	-205756.4947	-205235.2420
Pigs	-1673304.6600	-1673110.1193	-1672782.9907	**-1672522.5336**
BN_n100	**-328499.1000**	-334554.4000	-334223.4000	-333286.0000
BN_n200	**-667642.3000**	-682648.9000	-682081.4000	-680601.0000
		FastCHC$_{Mod2}$ ($\alpha = 40\%$)	FastCHC$_{Mod2}$ ($\alpha = 60\%$)	FastCHC$_{Mod2}$ ($\alpha = 80\%$)
Alarm		-48001.0012	-48001.0012	-48006.6924
Barley		-264254.3812	-266033.6374	-265699.4747
Hailfinder		-250346.7057	-250396.9589	-250472.7089
Insurance		**-67019.6496**	-67034.4770	-67058.0366
Mildew		-237832.5022	-243042.4006	-242635.9131
Munin1		-203452.6020	-204512.2889	-204837.3228
Pigs		-1673019.6149	-1672623.7133	-1672552.5238
BN_n100		-330526.2000	-331589.1000	-332992.9000
BN_n200		-673222.0000	-676501.1000	-678693.6000

Dataset	HC	FastCHC	FastCHC$_{Mod1}$ ($L = 2$)	FastCHC$_{Mod1}$ ($L = 3$)
Alarm	3444.2000	**1533.8000**	1673.4000	1848.0000
Barley	5680.8000	**1876.2000**	1993.4000	2123.8000
Hailfinder	7063.2000	**2401.0000**	2556.4000	2673.6000
Insurance	2230.0000	**1075.2000**	1266.0000	501.2000
Mildew	2460.4000	**906.4000**	960.0000	1041.4000
Munin1	102218.2000	**58598.8000**	62181.2000	67445.4000
Pigs	539248.6000	**125902.2000**	132401.0000	146048.2000
BN_n100	20712.2500	**6297.6000**	6712.9000	7658.3500
BN_n200	99766.6000	**22819.7500**	23591.9500	25775.7000
		FastCHC$_{Mod2}$ ($\alpha = 40\%$)	FastCHC$_{Mod2}$ ($\alpha = 60\%$)	FastCHC$_{Mod2}$ ($\alpha = 80\%$)
Alarm		2635.0000	2436.6000	2211.2000
Barley		3758.0000	3305.4000	2926.4000
Hailfinder		5031.2000	4427.2000	3698.8000
Insurance		1892.4000	1774.8000	1669.2000
Mildew		1695.8000	1418.8000	1217.8000
Munin1		94122.4000	91824.2000	88950.8000
Pigs		317561.4000	257212.0000	198098.6000
BN_n100		13831.8500	11059.7500	8753.0500
BN_n200		50421.9000	39880.6500	31093.2000

6 Conclusions

We have defined some modifications for constrained local search algorithms in order to obtain higher quality solutions closer to the state of the art algorithms when maintaining an efficient algorithm. The second modification proposed introduces a parameter which can be tuned in order to adjust the behaviour of the algorithm regarding the efficiency/quality tradeoff. Future works could lead to a parameters-free modification of the algorithm, in which the size of the FP sets

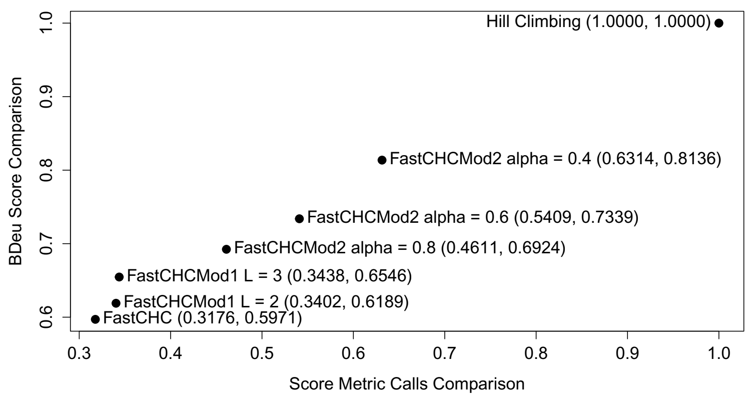

Fig. 2. Comparison between the different algorithms regarding score metric calls (x-axis) and score metric value (y-axis). The values displayed next to the algorithms names express the averaged ratios (calls, score) relative to the hill climbing algorithm from all datasets.

is reduced dynamically during the search process, trying to relax the constrains in the lasts steps of the search to take advantage of both constrained and unconstrained approaches. Preliminary experiments using a fixed rate reduction of the FP sets and statistical parameters such as the score variance between solutions have shown similar results to the parametrized modification.

Acknowledgments. This work has been partially funded by FEDER funds and the Spanish Government (MICINN) through project TIN2010-20900-C04-03.

References

1. Alonso-Barba, J.: delaOssa, L., Gámez, J., Puerta, J.: Scaling up the greedy equivalence search algorithm by constraining the search space of equivalence classes. International Journal of Approximate Reasoning (2013)
2. Chickering, D.M.: Learning bayesian networks is NP-complete. In: Learning from data, pp. 121–130. Springer (1996)
3. Gámez, J., Mateo, J., Puerta, J.: 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)
4. Gámez, J., Mateo, J., Puerta, J.: One iteration CHC algorithm for learning Bayesian networks: an effective and efficient algorithm for high dimensional problems. Progress in Artificial Intelligence 1(4), 329–346 (2012)
5. Gámez, J., Salmerón, A., Cano, A.: Design of new algorithms for probabilistic graphical models. Implementation in Elvira. Programo Research Project (TIN2007-67418-c03) (2010)

6. Heckerman, D., Geiger, D., Chickering, D.M.: Learning bayesian networks: The combination of knowledge and statistical data. Machine Learning 20(3) (1995)
7. Jensen, F.V., Nielsen, T.D.: Bayesian networks and decision graphs. Springer (2007)
8. Neapolitan, R.E.: Learning bayesian networks. Pearson Prentice Hall (2004)
9. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausble Inference. Morgan Kaufmann Pub. (1988)
10. Tsamardinos, I., Brown, L., Aliferis, C.: The max-min hill-climbing bayesian network structure learning algorithm. Machine Learning 65(1), 31–78 (2006)