Bayesian Network Structure Learning from Limited Datasets through Graph Evolution

Alberto Paolo Tonda[1], Evelyne Lutton[2],
Romain Reuillon[1], Giovanni Squillero[3], and Pierre-Henri Wuillemin[4]

[1] Institut des Systèmes Complexes, 57-59 rue Lhomond, 75005, Paris, France
[2] INRIA Saclay-Ile-de-France, AVIZ Team
LRI - Bâtiment 650, Université Paris-Sud, 91405, Orsay Cedex, France
[3] Politecnico di Torino, DAUIN, Corso Duca degli Abruzzi 124, 10129, Torino, Italy
[4] LIP6 - Département DÉSIR, 4, place Jussieu, 75005, Paris
{alberto.tonda,romain.reuillon}@iscpif.fr, evelyne.lutton@inria.fr,
giovanni.squillero@polito.it, pierrehenri.wuillemin@lip6.fr

Abstract. Bayesian networks are stochastic models, widely adopted to encode knowledge in several fields. One of the most interesting features of a Bayesian network is the possibility of learning its structure from a set of data, and subsequently use the resulting model to perform new predictions. Structure learning for such models is a NP-hard problem, for which the scientific community developed two main approaches: score-and-search metaheuristics, often evolutionary-based, and dependency-analysis deterministic algorithms, based on stochastic tests. State-of-the-art solutions have been presented in both domains, but all methodologies start from the assumption of having access to large sets of learning data available, often numbering thousands of samples. This is not the case for many real-world applications, especially in the food processing and research industry. This paper proposes an evolutionary approach to the Bayesian structure learning problem, specifically tailored for learning sets of limited size. Falling in the category of score-and-search techniques, the methodology exploits an evolutionary algorithm able to work directly on graph structures, previously used for assembly language generation, and a scoring function based on the Akaike Information Criterion, a well-studied metric of stochastic model performance. Experimental results show that the approach is able to outperform a state-of-the-art dependency-analysis algorithm, providing better models for small datasets.[1]

Keywords: Evolutionary computation, Bayesian network structure learning, Bayesian networks, Genetic Programming, Graph representation.

[1] Acknowledgments for the funding received from the European Community's Seventh Framework Programme (FP7/2009-2013) under grant agreement DREAM n. 222654-2.

A. Moraglio et al. (Eds.): EuroGP 2012, LNCS 7244, pp. 254–265, 2012.

1 Introduction

Bayesian networks are stochastic models widely used to encode knowledge in several different fields: computational biology and bioinformatics (gene regulatory networks, protein structure, gene expression analysis), medicine, document classification, information retrieval, image processing, data fusion, decision support systems, engineering, gaming and law.

A particularly interesting feature of a Bayesian network is the possibility of learning its structure from a set of data and subsequently use the obtained model to predict new results. However, the number of possible structures is superexponential in the number of variables of the model [1] and the problem of Bayesian network learning is proved to be NP-hard [2]. The machine learning community answered to this challenge with a research line dating back almost 30 years, giving birth to a class of effective deterministic algorithms that systematically determine the skeleton of the underlying graph and proceed to orient all arcs whose directionality is dictated by conditional independencies observed.

This interesting problem raised also the attention of the evolutionary computation community: several attempts at Bayesian network structure learning have been presented in recent years, ranging from cooperative coevolution [3] [4] [5] to evolutionary programming [6], to hybrid solution combining evolutionary approaches with heuristic search [7]. Both deterministic and evolutionary techniques share the assumption of the availability of dataset numbering thousands or hundreds of thousands of samples: for several real-world problems, however, this may not be the case [8]. For example, in the field of food processing and research, extremely time-consuming processes are required to get a small amount of sparse data.

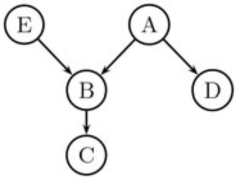
Taking inspiration from such a category of problems, this paper presents an evolutionary-based approach to Bayesian network structure learning, working with datasets of extremely reduced size. The proposed technique exploits a state-of-the-art evolutionary algorithm able to directly manipulate graph-like structures, previously used for assembly language generation, and makes use of a fitness function based on the Akaike information criterion, a metric taking into account both the accuracy and the complexity of a candidate model.

The rest of the paper is organized as follows. Section 2 briefly introduces the necessary background concepts on Bayesian networks. The proposed methodology is described in detail in section 3. Section 4 presents the case study chosen for the experimental evaluation, an established benchmark widely used in the Bayesian network learning field. Experimental results showing a comparison with a state-of-the-art dependency analysis algorithm are reported in section 5, while section 6 draws the conclusions and gives an outline for future works.

2 Background

2.1 Bayesian Networks

A Bayesian Network (BN) is a “graph-based model of a joint multivariate probability distribution that captures properties of conditional independence between



Node	Parents	Probabilities
A		$P(A=a_1) = 0.99$ $P(A=a_2) = 0.01$
B	A,E	$P(B=b_1 \mid A=a_1,E=e_1) = 0.5$ $P(B=b_2 \mid A=a_1,E=e_1) = 0.5$ $P(B=b_1 \mid A=a_1,E=e_2) = 0.1$ $P(B=b_2 \mid A=a_1,E=e_2) = 0.9$ $P(B=b_1 \mid A=a_2,E=e_1) = 0.4$ $P(B=b_2 \mid A=a_2,E=e_1) = 0.6$ $P(B=b_1 \mid A=a_2,E=e_2) = 0.2$ $P(B=b_2 \mid A=a_2,E=e_2) = 0.8$
C	B	$P(C=c_1 \mid B=b_1) = 0.3$ $P(C=c_2 \mid B=b_1) = 0.7$ $P(C=c_1 \mid B=b_2) = 0.5$ $P(C=c_2 \mid B=b_2) = 0.5$
D	A	$P(D=d_1 \mid A=a_1) = 0.8$ $P(D=d_2 \mid A=a_1) = 0.2$ $P(D=d_1 \mid A=a_2) = 0.7$ $P(D=d_2 \mid A=a_2) = 0.3$
E		$P(A=e_1) = 0.75$ $P(A=e_2) = 0.25$

Fig. 1. On the left, a directed acyclic graph. On the right, the parameters it is associated with. Together they form a Bayesian network BN whose joint probability distribution is $P(BN) = P(A)P(B|A,E)P(C|B)P(D|A)P(E)$.

variables" [9]. For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. The network could thus be used to compute the probabilities of the presence of various diseases, given the symptoms.

Formally, a Bayesian network is a directed acyclic graph (DAG) whose nodes represent variables, and whose edges encode conditional dependencies between the variables. This graph is called the *structure* of the network and the nodes containing probabilistic information are called the *parameters* of the network. Figure 1 reports an example of a Bayesian network.

The set of parent nodes of a node X_i is denoted by $pa(X_i)$. In a Bayesian network, the joint probability distribution of the node values can be written as the product of the local probability distribution of each node and its parents:

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i|pa(X_i))$$

2.2 Akaike Information Criterion

The Akaike information criterion (AIC) is a measure of the relative goodness of fit of a statistical model [10]. It is grounded in the concept of information entropy, in effect offering a relative measure of the information lost when a given model is used to describe reality. It can be said to describe the trade-off between bias and variance in model construction, or loosely speaking, between accuracy and dimension of the model. Given a data set, several candidate models may be ranked according to their AIC values: thus, AIC can be exploited as a metric for model selection.

When dealing with Bayesian networks, AIC is expressed as a composition of the loglikelihood, a measure of how well the candidate model fits the given dataset, and a penalty tied to the dimension of the model itself. The dimensional penalty is included because, on the one hand, the loglikelihood of a Bayesian network usually grows monotonically with the number of arcs, but on the other hand, an excessively complex network cannot be validated or even interpreted by a human expert. The loglikelihood of a model M given a dataset T is computed as

$$LL(M|T) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} log_2 \frac{N_{ijk}}{N_{ij}}$$

where n is the number of variables, r_i is the number of different values that the stochastic variable X_i can assume, q_i is the total number of possible configurations of its parent set $pa(X_i)$, N_{ijk} is the number of instances in the dataset T where the variable X_i takes its k-th value x_{ik} and the variables in $pa(X_i)$ take their j-th configuration w_{ij}, and N_{ij} is the number of instances in the dataset T where the variables in $pa(X_i)$ take their j-th configuration w_{ij}.

Taking for example the Bayesian network BN described in Figure 1, the loglikelihood of a dataset composed of one sample such as $T = (a_1, b_2, c_1, d_2, e_2)$ would be equal to

$$\begin{aligned} LL(BN|T) = log_2(P(A = a_1) \cdot P(B = b_2|A = a_1, E = e_2) \cdot \\ \cdot P(C = c_1|B = b_2) \cdot P(D = d_2|A = a_1) \cdot P(E = e_2)) = \\ = log_2(0.99 \cdot 0.9 \cdot 0.5 \cdot 0.3 \cdot 0.25) = -4.9 \end{aligned}$$

It is important to notice that datasets are usually composed by multiple samples, and that the final loglikelihood is the sum of the loglikelihoods of each sample.

Using the same formulation, the dimensional penalty of model M can be expressed as

$$|M| = \sum_{i=1}^{n} (r_i - 1) q_i$$

In the example of Figure 1, it would be thus:

$$|BN| = (1)_{penaltyA} + (4)_{penaltyB} + (2)_{penaltyC} + (2)_{penaltyD} + (1)_{penaltyE} = 10$$

In the canonical representation, the final AIC score is expressed as:

$$AIC = -2 \cdot (LL - |M|)$$

It is interesting to notice how the decomposability of the AIC makes it particularly feasible for use in a fitness function.

2.3 Bayesian Network Structure Learning

Learning the structure of a Bayesian network starting from a dataset is a NP-hard problem [2], that becomes more and more challenging as the size of the learning dataset decreases. The algorithmic approaches devised to solve this problem can be divided into two main branches: score-and-search heuristics and deterministic algorithms that rely upon statistical considerations on the learning set.

In recent years, Bayesian network structure learning gained more and more attention from the evolutionary community. Several attempts to tackle the problem has been tested, ranging from evolutionary programming [6], to hybrid approaches [7], to cooperative coevolution [3] [4].

The underlying assumption of these works is often the availability of a considerable number of samples in the dataset used for learning: under these conditions, however, a class of deterministic algorithms known as dependency analysis is able to deliver results of high quality in a fraction of the time. One of the most performing algorithms in this category is known as *Greedy Thick Thinning* (GTT)[11]. Starting from a completely connected graph, first GTT applies the well-known PC algorithm [12], that cuts arcs on the basis of conditional independence tests; then, it starts first adding and then removing arcs, scoring the network after each modification and using a set of heuristics to avoid a premature convergence. GTT implementations can be found in products such as GeNie/SMILE [13].

When the number of samples available for structure learning is small, however, dependency analysis algorithms see their performance degrade dramatically, leaving a promising open area of applicability for evolutionary techniques.

3 Proposed Methodology

The proposed approach to Bayesian network structure learning belongs to the category of score-and-search techniques: the evolutionary core is a state-of-the-art evolutionary algorithm, while the scoring metric used is a decomposition of the AIC.

3.1 μGP

μGP^3 [14] is an evolutionary algorithm tool developed by the CAD Group of Politecnico di Torino and available as a GPL software [15].

The main difference between μGP^3 and the classical Genetic Programming paradigm [16], is the encoding of individuals in tagged graphs instead of trees. More precisely, the algorithm makes use of constrained tagged graphs, that is, directed graphs where every element may own one or more tags, and that in addition have to respect a set of constraints. A tag is a name-value pair whose purpose is to convey additional information about the element it belongs to. Tags are used to add semantic information to graphs, augmenting the nodes with a number of parameters, and also to uniquely identify each element during the evolution. The constraints may affect both the information contained in the graph elements and its structure. Graphs are modified by genetic operators, such as the classical mutation and recombination, but also by different operators, as required. The activation probability and strength for every operator is an endogenous parameter. The genotype of every individual is described by one or more constrained tagged graphs, each of which is composed by one or more sections. Sections allow defining a global structure for the individuals that closely follows the structure of any candidate solution for the problem.

Constraints limit the possible productions of the evolutionary tool, and also provide them with semantic value. A user-defined XML configuration file encodes the constraints, which in turn provide the genotype-phenotype mapping for the generated individuals, describe their possible structure and define which values the existing parameters (if any) can take.

Individuals' fitness is computed by means of an external evaluator: this is usually a script that runs a simulation using the individual as input and collects the results, but may be any program able to provide the evolutionary core with proper feedback. The fitness of an individual is represented by a sequence of floating point numbers optionally followed by a comment string. This is currently used in a prioritized fashion: one fitness A is considered greater than another fitness B if the nth component of A is greater than the n-th component of B and all previous components (if any) are equal; if all components are equal then the two fitnesses are considered equal.

Given the versatility and the ease of configuration of the tool, it is not surprising that several successful applications of μGP^3 appear in literature, mainly in the area of assembly language generation [17], but also in very different fields, such as software verification [18].

The ability of evolving directed graphs, with genetic operators already designed to work on them, is particularly promising when dealing with problems where the individual is in fact a graph, as in the case of Bayesian network structure learning. A specific type of parameter, `innerBackwardLabel`, used in assembly generation for jumps to previous lines of the code, is now exploited to describe an oriented arc coming from a previous node. Other features of the algorithm are of particular use in this situation: a simple self-adapting mechanism avoids the need of setting fixed values for the genetic operators, increasing the activation probability for operators that constantly produce good offspring. A map of the individuals at genotype-level is also used to remove possible clones before the evaluation step.

The possibility of choosing the categories of genetic operators to apply is also advantageous: for this experience, two kinds of `crossover` and three `mutations` have been selected, as shown in Table 1. The difference between the crossovers is in the number of points of cut (one-point and two-point). The chosen mutations, operating on `innerBackwardLabel` type parameters, can collectively perform the addition, removal or variation of directed arcs in the graph. It is interesting to notice how the same mutation operators, applied to parameters such as floating point numbers, behave in a different fashion, changing the real-valued parameter according to a Gaussian distribution. Figure 2 shows an example of the effect of the `onePointCrossover` genetic operator on two individuals containing directed arcs.

3.2 Individual Encoding

A candidate solution to the Bayesian network structure learning problem is a directed acyclic graph. Most evolutionary approaches take into account the possibility of generating loops, by either using repairing operators or discarding

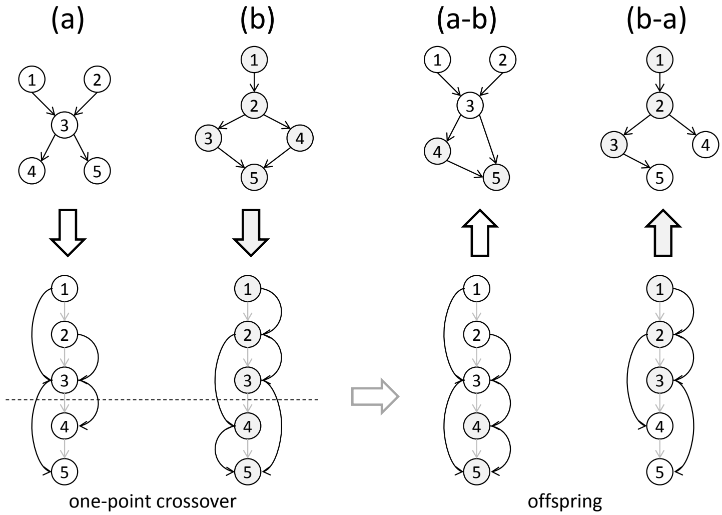


Fig. 2. Example of `onePointCrossover` application in μGP^3. When the genotype is of fixed length, the directed arcs are reattached to corresponding parts of the new individuals.

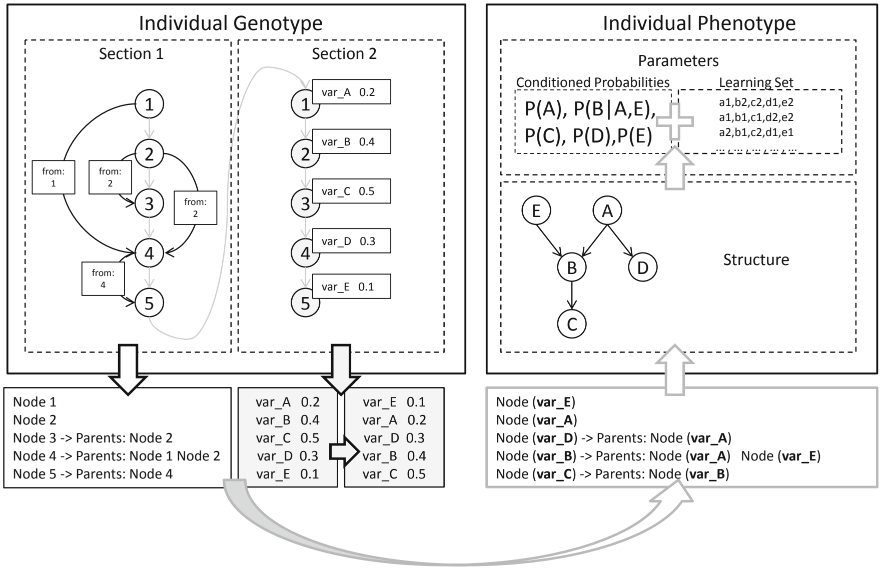


Fig. 3. Example of individual encoding in the proposed approach. The first part of the genome (Section 1) describes a directed acyclic graph, correct by construction. The second part (Section 2) specifies the mapping of the variables in the dataset on the nodes of the graph. The resulting phenotype is a graph with the structure of the first part of the genotype and the node ordering of the second. The parameters of the resulting Bayesian network are computed on the learning set, starting from the conditional probabilities derived from the structure.

graphs containing cycles. Even detecting the presence of a cycle, however, is non-trivial and computationally expensive.

Thanks to the options available in μGP^3, it is in fact possible to set the evolutionary algorithm to always generate valid structures. The nodes of the graph, in the individual description, will appear in a given order: it is sufficient to constraint the generation of directed arcs from one node to nodes that only appear after it in the individual description, and loops are automatically avoided by construction.

The mapping from the variables that appear in the learning dataset to the directed acyclic graph is encoded in the second part of the individual. Each variable is associated with a floating point weight ranging from 0 to 1: when the individual is evaluated, the variables are sorted with their weight and are subsequently mapped in-order to the graph structure described in the first part. An example of individual is reported in Figure 3.

The maximum number of parents for a single node is set to 10, as it is common for most search-and-score metaheuristics [3].

3.3 Fitness Function

Preliminary experiments with the canonical AIC scoring show a tendency for the algorithm to explore prevalently local optima of very simple structures: probably, in the AIC fitness landscape, the slope towards low complexity is much steeper than the one towards good loglikelihood values.

To avoid premature convergences, the fitness function used in the experiments is thus a hierarchical decomposition of the AIC. First, the loglikelihood of the model with respect to the learning dataset is considered; if two candidate models have the same loglikelihood, they are then compared on the penalty tied to their respective dimension.

4 Case Study

Over the course of time, the Bayesian network structure learning community developed a vast number of benchmarks, from simple to relatively complex. To assess the proposed approach, the Alarm network [19] is selected. Alarm was constructed for monitoring patients in intensive care, and it is effectively used. It features 37 nodes and 46 edges, and its overview can be found for example at `http://www.norsys.com/netlib/alarm.htm`

5 Experimental Results and Discussion

Experts usually agree that a comprehensive learning set for a Bayesian network should include at least 10-15 samples for each parameter. In the case of Alarm, this would mean the availability of thousands of samples: since the proposed approach is aimed at working with limited information, datasets smaller by an order of magnitude are selected.

Table 1. Parameters used for μGP^3 in all the performed experiments. λ is the number of genetic operators applied at each step. The `stagnation` stop condition forces the end of the execution if the best individual in the population does not change for a specified number of generations. For further details on the parameters, see [14].

Parameter	**Value**
μ	1,000
λ	1,000
Selection	`tournamentSelection` with τ = (1,4)
Diversity preservation	`fitnessHole` (0.5)
Stop condition	`stagnation` (100 generations)
Genetic operators	`singlePointCrossover`, `twoPointCrossover`, `alteratioMutation`, `replacementMutation`, `singleParameterAlteratioMutation`

Thus, to assess the validity of the methodology, three learning sets of 100 samples each are generated starting from the Alarm network description available in literature [19]. `alarm-100-a` is produced using the standard methodology of constrained probability generation; `alarm-100-b` contains 100 samples randomly taken from a dataset of original size 5,000; and `alarm-100-c` contains 100 samples randomly selected from a dataset of original size 10,000. It is important to note that, albeit randomly created, each 100-samples dataset contains all possible values for each variable in the Alarm network. In principle, all the datasets should have the same probability of being representative of the original network.

μGP^3 is set with the parameters reported in Table 1. Note that, since the algorithm is self-adapting, it is not necessary to initially select an activation probability for the genetic operators. The strenght of the genetic operators is also self-adapted during the evolution, with an initial value of $\sigma = 0.9$.

For each learning set (`alarm-100-a`, `alarm-100-b`, `alarm-100-c`), 20 runs of the proposed approach are executed, along with a run of *Greedy Thick Thinning*. Since the latter is deterministic, it reports the same output even for repeated runs.

Results are reported in Table 2. The first four lines show the performance of the true structure of the Alarm network, for comparison. It is interesting to notice the significant difference in loglikelihood values between the same structure, when the parameters are learned from different datasets (lines 2-4). Even knowing the true structure of the network, a dataset of considerable size would be needed to learn the true values for all the parameters. Learning algorithms of all categories can only compute an approximation of the parameters, that degrades with the reduction in size of the learning set. The grayed cells show the dataset the method is trained on.

The first evidence is that all solutions found by the proposed approach have lower dimensional penalties, much closer to the true structure than the ones delivered by GTT. While the loglikelihood values seem to favor GTT, it becomes evident that the greedy algorithm overfits the learning data: the networks delivered by GTT show loglikelihoods that are higher than the corresponding values of the original structure with parameters learned from the same dataset.

Table 2. Results of the experiments. While the logliklihood of solutions obtained with *Greedy Thick Thinning* is generally higher, it shows more overfitting when compared to the loglikelihood of the true structure with parameters trained on the same dataset, due to the higher dimension of the networks found.

Method	Dimensional Penalty		Loglikelihood `alarm-100-a`		Loglikelihood `alarm-100-b`		Loglikelihood `alarm-100-c`	
	Avg	StDev	Avg	StDev	Avg	StDev	Avg	StDev
True network	509.0	-	-1,571.90	-	-1,476.43	-	-1,470.95	-
True structure (`a`)	509.0	-	-1,651.05	-	-1,693.29	-	-1,672.12	-
True structure (`b`)	509.0	-	-1,807.32	-	-1,567.74	-	-1,688.81	-
True structure (`c`)	509.0	-	-1,809.77	-	-1,720.54	-	-1,569.26	-
`GTT-100a`	796.0	-	-1,641.61	-	-1,808.63	-	-1,801.35	-
`GTT-100b`	1,226.0	-	-1,979.28	-	-1,494.18	-	-1,825.90	-
`GTT-100c`	1,238.0	-	-1,918.89	-	-1,869.84	-	-1,498.90	-
μ`gp-100a`	763.39	42.54	-1,644.68	30.39	-1,861.99	34.06	-1,848.50	31.06
μ`gp-100b`	702.29	95.18	-1,981.79	27.47	-1,529.00	18.94	-1,850.80	23.89
μ`gp-100c`	669.00	96.21	-1,986.72	33.40	-1,873.79	26.82	-1,554.09	10.68

When dealing with such a small amount of data, all stochastic considerations that are at the base of the class of deterministic algorithms GTT belongs to, simply do not hold anymore: even relatively effective statistics tests yield wrong results. In such a difficult situation, an effective score-and-search metaheuristic can provide better results.

For a final remark on computational times, as anticipated in Subsection 2.3, a run of GTT lasts a few seconds. A single generation of the proposed approach, on the same machine, takes between 1 and 2 minutes to complete, with the global time for the whole process amounting to hours. In less than 5% of the runs, the algorithm is also restarted, after delving deep into parts of the fitness landscape with high dimension, thus increasing the evaluation time over an acceptable threshold. Since the main focus of the present work is not on efficiency, it must be noted that previous computations of conditional probability statements are not stored and reused; the possibility of parallel evaluations of individuals in the same generation is also not exploited.

6 Conclusions

Bayesian network structure learning from a dataset is a complex problem, especially when the learning set is small: but in many real-world problems, particularly in the food processing and research industry, it is possible to have access to limited-size datasets only.

This paper presents an approach to Bayesian network structure learning from datasets of limited size. The methodology is based on an evolutionary algorithm able to evolve directed graphs, that can be easily set to always produce acyclic graphs by construction, avoiding the repairing of non-valid structures.

The proposed methodology is assessed on a well-known benchmark, the Alarm network, and compared against a state-of-the-art dependency analysis algorithm, known as *Greedy Thick Thinning*. The experimental results show that the approach is effective, obtaining simpler structures, for which interpretation and validation by humans are more feasible. The proposed approach also avoids the likelihood overfitting on the learning set of the greedy algorithm.

Future works will compare the proposed approach with other evolutionary solutions in literature on small datasets. The effect of different metrics or combination of metrics, such as Maximum Description Length or Bayesian Information Criterion, on the scoring of the candidate solutions will be also explored, along with the possibility of using cooperative coevolution to split the original task in sub-problems.

References

1. Robinson, R.: Counting unlabeled acyclic digraphs. In: Little, C. (ed.) Combinatorial Mathematics V. Lecture Notes in Mathematics, vol. 622, pp. 28–43. Springer, Heidelberg (1977), doi:10.1007/BFb0069178
2. Chickering, D.M., Geiger, D., Heckerman, D.: Learning bayesian networks is np-hard. Technical Report MSR-TR-94-17, Microsoft Research, Redmond, WA, USA (November 1994)
3. Carvalho, A.: A cooperative coevolutionary genetic algorithm for learning bayesian network structures. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 1131–1138. ACM, New York (2011)
4. Wong, M.L., Lee, S.Y., Leung, K.S.: Data mining of bayesian networks using cooperative coevolution. Decis. Support Syst. 38, 451–472 (2004)
5. Barriere, O., Lutton, E., Wuillemin, P.H.: Bayesian network structure learning using cooperative coevolution. In: Genetic and Evolutionary Computation Conference, GECCO 2009 (2009)
6. Wong, M.L., Lam, W., Leung, K.S.: Using evolutionary programming and minimum description length principle for data mining of bayesian networks. IEEE Transactions on Pattern Analysis and Machine Intelligence 21(2), 174–178 (1999)
7. Fournier, F., Wu, Y., McCall, J., Petrovski, A., Barclay, P.: Application of evolutionary algorithms to learning evolved bayesian network models of rig operations in the gulf of mexico. In: 2010 UK Workshop on Computational Intelligence (UKCI), pp. 1–6 (September 2010)
8. Barrière, O., Lutton, E., Baudrit, C., Sicard, M., Pinaud, B., Perrot, N.: Modeling Human Expertise on a Cheese Ripening Industrial Process Using GP. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 859–868. Springer, Heidelberg (2008)
9. Friedman, N., Linial, M., Nachman, I., Pe'er, D.: Using bayesian networks to analyze expression data. In: Proceedings of the Fourth Annual International Conference on Computational Molecular Biology, RECOMB 2000, pp. 127–135. ACM, New York (2000)
10. Akaike, H.: A new look at the statistical model identification. IEEE Transactions on Automatic Control 19(6), 716–723 (1974)
11. Cheng, J., Bell, D.A., Liu, W.: An algorithm for bayesian belief network construction from data. In: Proceedings of AI & STAT 1997, pp. 83–90 (1997)

12. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search, 2nd edn. MIT Press Books, vol. 1. The MIT Press (2001)
13. Druzdzel, M.J.: SMILE: Structural modeling, inference, and learning engine and GeNIe: A development environment for graphical decision-theoretic models, pp. 902–903. American Association for Artificial Intelligence (1999)
14. Sanchez, E., Schillaci, M., Squillero, G.: Evolutionary Optimization: the uGP toolkit. Springer, Heidelberg (2011)
15. SourceForge: Host of μgp3, `http://sourceforge.net/projects/ugp3`
16. Koza, J., Poli, R.: Genetic programming. In: Burke, E.K., Kendall, G. (eds.) Search Methodologies, pp. 127–164. Springer, US (2005), doi:10.1007/0-387-28356-0_5
17. Squillero, G.: Microgp - an evolutionary assembly program generator. Genetic Programming and Evolvable Machines 6, 247–263 (2005)
18. Gandini, S., Ruzzarin, W., Sanchez, E., Squillero, G., Tonda, A.: A framework for automated detection of power-related software errors in industrial verification processes. J. Electron. Test. 26, 689–697 (2010)
19. Beinlich, I.A., Suermondt, H.J., Chavez, R.M., Cooper, G.F.: The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks. In: Second European Conference on Artificial Intelligence in Medicine, London, Great Britain, vol. 38, pp. 247–256. Springer, Berlin (1989)