Grammar-Guided Evolutionary Construction of Bayesian Networks

José M. Font, Daniel Manrique, and Eduardo Pascua

Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid. Campus de Montegancedo, 28660 Boadilla del Monte, Spain
jm.font@upm.es, dmanrique@fi.upm.es, e.pascua@alumnos.upm.es

Abstract. This paper proposes the EvoBANE system. EvoBANE automatically generates Bayesian networks for solving special-purpose problems. EvoBANE evolves a population of individuals that codify Bayesian networks until it finds near optimal individual that solves a given classification problem. EvoBANE has the flexibility to modify the constraints that condition the solution search space, self-adapting to the specifications of the problem to be solved. The system extends the GGEAS architecture. GGEAS is a general-purpose grammar-guided evolutionary automatic system, whose modular structure favors its application to the automatic construction of intelligent systems. EvoBANE has been applied to two classification benchmark datasets belonging to different application domains, and statistically compared with a genetic algorithm performing the same tasks. Results show that the proposed system performed better, as it manages different complexity constraints in order to find the simplest solution that best solves every problem.

Keywords: Evolutionary computation, Bayesian network, grammar-guided genetic programming.

1 Introduction

Bayesian networks (BN) are computational tools that can perform probabistic inference from data with uncertainty [1]. They have been applied as an automatic reasoning mechanism to a wide range of domains [2,3]. A BN is a directed acyclic graph (DAG) that codifies the existing dependencies between its nodes, each of what contains a conditional probability table [4]. The automatic learning of a BN from data is a two-step procedure composed of the design of the network topology and the calculation of its conditional probability tables [5]. This has been proven to be an NP-Hard procedure [6]. For this reason, knowledge engineering techniques need to be used to achieve quality solutions [7].

Evolutionary computation has been successfully applied to solve search and optimization problems, such as the generation of both symbolic and sub-symbolic self-adapting intelligent systems [8]. Its application to the automatic construction of BN must overcome several difficulties, such as the design of an accurate codification system that can manage acyclic graphs and the implementation of

J.M. Ferrández et al. (Eds.): IWINAC 2011, Part I, LNCS 6686, pp. 60–69, 2011.

genetic operators that prevent the generation of illegal graph structures. Insofar as genetic algorithms do not solve the closure problem [9], crossover and mutation operators may generate invalid individuals. For this reason, a repair operator must be executed in order to transform invalid structures into DAGs [10].

Different evolutionary approaches codify BNs into an array of nodes that can only be forward connected [11,12,13]. In order to preserve DAG properties, crossover and mutation operators perform order permutations over this array only. The system presented in [14] replaces the crossover operator with several specific mutation operators that perform controlled mutations on the genome of the individuals. The work described in [15] implements a restricted crossover operator that reduces the search space but always produces valid offspring. In the same way, the system in [16] reduces the search space by only allowing the generation of naive-Bayes classifiers, whose straightforward codification decreases the complexity of the evolutionary procedure and avoids the need for preserving DAG structures.

This paper presents the EvoBANE (Evolutionary Bayesian Networks) system, a grammar-guided evolutionary system for automatically generating Bayesian networks that solve classification problems. EvoBANE implements a context-free grammar (CFG) generator and a fitness calculator module. The CFG generator inputs the specifications of the application domain, as well as the features of the solutions to be built. It then outputs the CFG that generates the language codifying the solution space of all the valid BN structures (individuals) for those specifications and features. The fitness calculator first calculates the conditional probability tables of the individuals by means of a probabilistic estimator [17], and then evaluates the individual's accuracy as an instance classifier. EvoBANE is based on GGEAS technology [18] to initialize and evolve a population of BN structures. Evolution is achieved by means of a general-purpose grammatical crossover operator [19] that avoids the closure problem without including a repair operator. This way, EvoBANE always generates valid individuals, preserving the DAG properties without using constraints that prevent it from exploring the whole solution space. The efficiency of EvoBANE for building Bayesian classifiers has been tested in two different application domains. A genetic algorithm with a single-point crossover operator was run on the same problems, and the results were compared.

2 The EvoBANE System

EvoBANE's structure is an extension of the modular design implemented in GGEAS. It consists of two independent components: a grammar-guided genetic programming (GGGP) core and an external layer. The GGGP core is common to every GGEAS implementation and remains unchanged whatever the chosen application domain. The external layer is composed, in this case, of two special-purpose modules: the CFG generator and the fitness calculator, whose implementation directly depends on the GGEAS application domain. EvoBANE

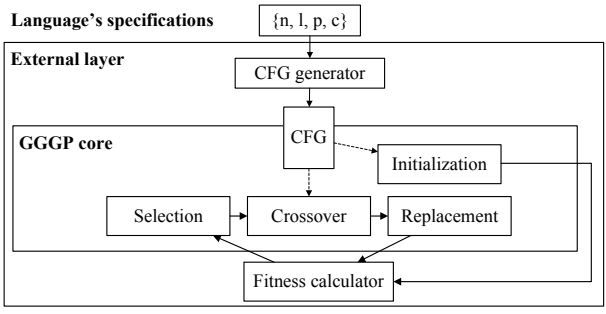

Fig. 1. Components and execution flow of EvoBANE

adopts the GGGP core and builds its own external layer modules in order to compose a system that automatically generates Bayesian networks that solve classification problems.

Figure 1 displays the execution flow of EvoBANE over the different components of its architecture. The CFG generator receives the specifications of the language that defines the solution space and outputs the context-free grammar that generates that language. This grammar is used by the GGGP core to initialize a population of BN structures. They are first evaluated by the fitness calculator module and then reinserted into the GGGP core for selection, crossover and replacement. The following sections explain both external modules in detail.

2.1 The CFG Generator

A context-free grammar is defined as a 4-tuple $G = (\Sigma_N, \Sigma_T, S, P) / \Sigma_N \cap \Sigma_T = \emptyset$, where Σ_N is the alphabet of non-terminal symbols, Σ_T is the alphabet of terminal symbols, S is the axiom and P is the set of production rules written in Backus-Naur form. The CFG generator in EvoBANE automatically builds grammars of this kind whose production rules generate individuals that codify valid BN structures. For this purpose, the CFG generator needs four input parameters: n, the number of nodes of the networks to be codified; l, the number of layers in which those nodes are distributed; p, the maximum permitted number of input connections per node and c, the maximum permitted number of input connections for the class node.

The number of nodes $n \in \mathbb{N}$ is extracted from the application domain and equals the number of feature variables observed for classifying an instance set. The class node that contains the possible classes in which instances are classified is counted as an extra node. Consequently, the total number of nodes is $n + 1$. Nodes are distributed throughout a list of l layers, where $l \in \mathbb{N}$, $1 \leq l \leq n$, and the class node alone is stored in the last layer. A layer is an array that contains a set of nodes arranged in a fixed order. Layers can be reordered to generate different BN structures, but the position of the nodes within a layer must be unchanged. Nodes arranged in layers are forward connected to each

other (even within the same layer). No backward connections are allowed in order to preserve DAG properties. Connections between nodes represent their conditional dependencies. Every node has a limited number of input connections p, where $p \in \mathbb{N}$, $0 \leq p \leq n-1$, except for the class node whose limit equals c, so that $c \in \mathbb{N}$, $1 \leq c \leq n$.

Grammars generated by the CFG generator are named after the four parameters n, l, p and c in the form G_{nlpc}. These four parameters vest the CFG generator of EvoBANE with the flexibility to modify the search space of a problem depending on its complexity (given by the value of n), allowing the evolutionary system to find the simplest BN structure that best solves that problem. Insofar as the value of n is fixed, the lower the value of l, p and c is, the smaller the combinatorial explosion generated from reordering and connecting the nodes will be. Notice that a grammar created with $l = n$, $p = n - 1$ and $c = n$, generates the language of all the valid BN structures for a given n. On the other hand, a grammar with $l = n$, $p = 0$ (complete conditional independence), and $c = n$ generates the language of all the valid naive-Bayes structures for a given n.

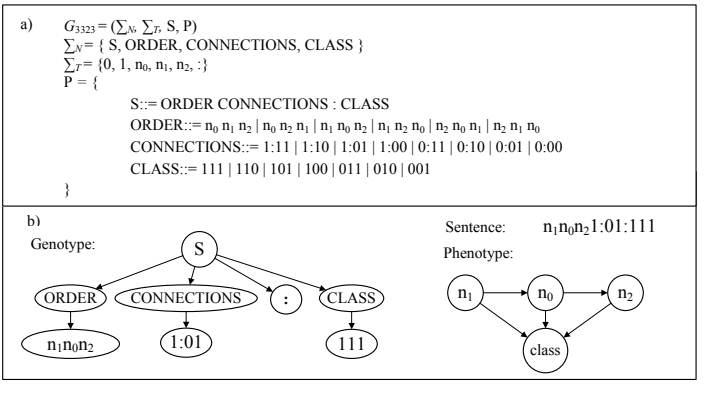

Fig. 2. a) Sample grammar G_{3323} automatically generated by the CFG generator. b) Sample individual that belongs to the language generated by G_{3323}.

Figure 2.a shows the sample grammar G_{3323} generated by the CFG generator for the values $n = 3$, $l = 3$, $p = 2$ and $c = 3$. Every individual generated by the grammar is composed of three parts: ORDER, CONNECTIONS and CLASS. The first part references to the order in which the layers of nodes are arranged. Insofar as the number of nodes and layers is the same in this example($n = l$), every layer contains only one node. The second part refers to the list of input connections of the n nodes in the individual, and the last part represents the input connections of the class node. The set of terminal symbols stores one symbol per node (except for the class node), a divider token ":" and numbers "1" and "0", which codify the presence or absence of a connection, respectively.

The first of the production rules displays the sequence of the three components of an individual. The second production rule determines an ordered list of nodes

in the individual. The cardinality of the set of possible configurations of layers equals $l!$, which, in this example, is equal to $n!$ insofar as $n = l$. The third production rule codifies the input connections of every position in the ordered list separated by the divider token. Notice that the node occupying the first position in the list always receives zero input connections, as nodes can only be forward connected. For this reason, the codified connections start at the second position in the list, which can only receive one input connection from the first position. The second codified set of connections, after the divider token, refers to the second position in the list of nodes, which can receive input connections from positions one and two. The last production rule establishes that the class node may be connected with $c = 3$ nodes. Notice that connections refer to positions in the list of nodes, not actual nodes.

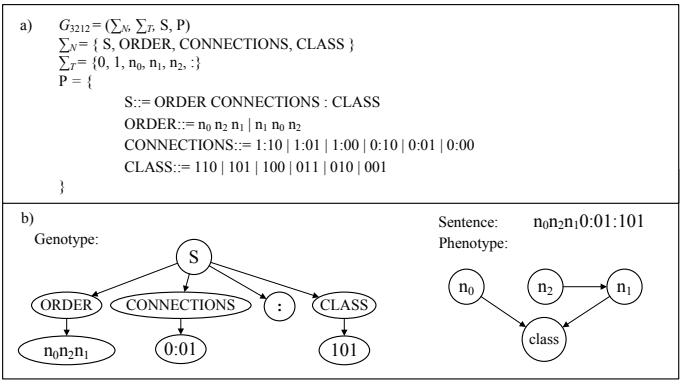

Fig. 3. a) Sample grammar G_{3212} automatically generated by the CFG generator. b) Sample individual that belongs to the language generated by G_{3212} .

Figure 2.b shows an individual that belongs to the language generated by G_{3323}. The derivation tree shown on the left is built by choosing the third consequent produced by "ORDER", that establishes the ordered list n_1, n_0, n_2. Then the third consequent produced by "CONNECTIONS" is chosen, codifying one input connection for the second node (n_0) coming from the first node (n_1), and another one for the third node (n_2) coming from the second node. Finally, the first consequent produced by "CLASS" fully connects the class node with n_0, n_1 and n_2. This derivation tree is the genotype that codifies the sentence shown in the upper-right of the diagram. The sentence codifies the phenotype of the individual, shown in the lower-right of the diagram.

Figure 3.a shows another sample grammar G_{3212} generated for the values $n = 3$, $l = 2$, $p = 1$ and $c = 2$. This grammar follows the same structure as presented in Figure 2.a. In this example, nodes ($n = 3$) are alternatively distributed across two layers ($l = 2$): $layer\,1 = \{n_0, n_2\}$ and $layer\,2 = \{n_1\}$. Insofar as only layers are reordered, the second rule produces only $\{n_0, n_2, n_1\}$ and $\{n_1, n_0, n_2\}$, a total of $l!$ orders. The third and fourth production rules limit

the number of input connections to $p = 1$ for variable nodes, and to $c = 2$ for the class node, respectively. Like Figure 2.b, Figure 3.b shows the genotype, sentence and phenotype of an individual belonging to the language generated by G_{3212}.

The grammars in Figures 2.a and 3.a can be used to generate BN structures that solve classification problems with $n = 3$ feature variables. They differ in that the language generated by G_{3212} builds a smaller search space than the generated by G_{3323}, lowering the complexity of EvoBANE's evolutionary process.

2.2 The Fitness Calculator

Every individual is assigned a fitness score when it first enters the fitness calculator module. This score is a percentage measure that represents how accurately the BN codified by the individual (derivation tree) classifies a training set of instances. First, the fitness calculator decodifies the individual to output its BN structure (phenotype). Then, for each node in the network, a probabilistic estimator included in the Weka framework estimates its conditional probability table (CPT) [17]. Briefly, this estimator estimates the probability of every value in the node by counting the number of occurrences of that value within the training set. Once all the CPTs have been estimated, the fitness calculator uses the BN to classify the instances in the training set and calculates the fitness of the individual as the percentage of correctly classified instances.

3 Results

EvoBANE was used to classiffy two different datasets that belong to two different application domains extracted from the UCI repository [20]. The first one is called "Vote" and classifies voters as "Republicans" or "Democrats" considering sixteen feature variables. For this dataset, the CFG generator was set to generate the grammars G_{16433}, G_{16333}, and G_{16133}, to tackle the problem from three different angles. A genetic algorithm (GA) with a repair mechanism [17] was added as a fourth approach to compare its performance with EvoBANE. These four approaches were executed 20 times. Each execution is set up to evolve a population of 10 individuals for 100 generations.

Table 1 shows the descriptive statistics of the results of the four approaches in both training and test sets. The mean column shows that EvoBANE provides the best approaches in both the training and testing phases. The standard deviation column indicates that the GA always gets the same maximum fitness. This fitness is under the lower bound of the 95% confidence interval for the mean of the three EvoBANE approaches in both the training and testing phases. One possible explanation is that the genetic algorithm gets trapped in a local optimum with a fitness score equal to 92.7586, and is unable to explore the solution space as EvoBANE does.

An analysis of variance (ANOVA) test was performed in order to statistically compare the mean fitness of each of the three EvoBANE approaches. Table 2 details the results of the ANOVA test, where the null hypotheses (mean fitness

Table 1. Descriptive statistical results for 20 executions of EvoBANE with grammars $G_{16\,4\,3\,3}$, $G_{16\,3\,3\,3}$, $G_{16\,1\,3\,3}$ and a genetic algorithm for the "Vote" dataset

		N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Minimum	Maximum
						Lower Bound	Upper Bound		
Fitness training	$G_{16\,4\,3\,3}$	20	95.8966	.2209	.0494	95.7931	96.0000	95.5172	96.2069
	$G_{16\,3\,3\,3}$	20	95.9828	.2313	.0517	95.8745	96.0910	95.5172	96.2069
	$G_{16\,1\,3\,3}$	20	96.1207	.6216	.1390	95.8298	96.4116	95.5172	98.6207
	GA	20	92.7586	.0000	.0000	92.7586	92.7586	92.7586	92.7586
	Total	80	95.1897	1.4557	.1627	94.8657	95.5136	92.7586	98.6207
Fitness testing	$G_{16\,4\,3\,3}$	20	96.7586	.5526	.1236	96.5000	97.0173	95.8621	97.2414
	$G_{16\,3\,3\,3}$	20	96.5517	.5003	.1119	96.3176	96.7859	95.8621	97.2414
	$G_{16\,1\,3\,3}$	20	96.3448	.9253	.2069	95.9118	96.7779	93.1034	97.2414
	GA	20	95.8621	.0000	.0000	95.8621	95.8621	95.8621	95.8621
	Total	80	96.3793	.6720	.0751	96.2298	96.5289	93.1034	97.2414

Table 2. Results of the ANOVA test for the "Vote" dataset

		Sum of Squares	df	Mean Square	F	Sig.
Fitness training	Between Groups	.511	2	.256	1.569	.217
	Within Groups	9.287	57	.163		
	Total	9.798	59			
Fitness testing	Between Groups	1.712	2	.856	1.819	.171
	Within Groups	26.825	57	.471		
	Total	28.537	59			

values are equal in the training and test sets) cannot be rejected with a significance of the value $F(df = 2/57)$ equal to 0.217 in training and 0.171 in the testing. Even though the null hypotheses cannot be rejected, the significance values indicate that there is a high probability of finding differences between the means. Tukey's HSD test has been used to show where the differences between the three approaches lie.

Table 3 contains the results of Tukey's HSD test. It shows that the means of the first two approaches are closer to each other than to the third approach in both the training and testing phases. Insofar as the lowest fitness during the testing phase belongs to the third approach, we infer that the simplification of the search space by the grammar $G_{16\,1\,3\,3}$ leads to slight drop in the accuracy of the BNs generated by EvoBANE.

The second dataset is called "Postoperative", and its classification task is to determine where patients in a postoperative recovery area should be sent to next: intensive care unit, hospital floor or home. Eight feature variables should be used for classification purposes. The CFG generator was set to generate the grammars G_{8444}, G_{8344} and G_{8144}. Table 4 shows the descriptive statistics of these three EvoBANE approaches and the GA. All the EvoBANE approaches again achieve better results than the GA in both the training and the testing phases.

Table 5 details the results of the ANOVA test performed on the three EvoBANE approaches. In this dataset, the equality of the means (null hypothesis) cannot be rejected in the testing phase with a significance of the value $F(df = 2/57)$ equal to 0.886. This suggests that all three EvoBANE approaches achieve the same results.

Table 3. Results of Tukey's HSD post-hoc test for the "Vote" dataset

Dependent Variable	(I) Approach	(J) Approach	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
						Lower Bound	Upper Bound
Fitness training	G_{16433}	G_{16333}	-.08621	.1105	.863	-.3766	.2042
		G_{16133}	-.22414	.1105	.187	-.5145	.0662
	G_{16333}	G_{16433}	.08621	.1105	.863	-.2042	.3766
		G_{16133}	-.13793	.1105	.599	-.4283	.1524
	G_{16133}	G_{16433}	.22414	.1105	.187	-.0662	.5145
		G_{16333}	.13793	.1105	.599	-.1524	.4283
Fitness testing	G_{16433}	G_{16333}	.20690	.1879	.690	-.2866	.7004
		G_{16133}	.41379	.1879	.132	-.0797	.9073
	G_{16333}	G_{16433}	-.20690	.1879	.690	-.7004	.2866
		G_{16133}	.20690	.1879	.690	-.2866	.7004
	G_{16133}	G_{16433}	-.41379	.1879	.132	-.9073	.0797
		G_{16333}	-.20690	.1879	.690	-.7004	.2866

Table 4. Descriptive statistical results for 20 executions of EvoBANE with grammars G_{8444}, G_{8344}, G_{8144} and a genetic algorithm for the "Postoperative" dataset

		N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Minimum	Maximum
						Lower Bound	Upper Bound		
Fitness training	G_{8444}	20	80.00	.000	.000	80.00	80.00	80	80
	G_{8344}	20	81.50	4.617	1.032	79.34	83.66	80	95
	G_{8144}	20	80.00	.000	.000	80.00	80.00	80	80
	GA	20	75.00	.000	.000	75.00	75.00	75	75
	Total	80	79.13	3.354	.375	78.38	79.87	75	95
Fitness testing	G_{8444}	20	72.6667	6.3614	1.4225	69.6894	75.6439	63.3333	80
	G_{8344}	20	73.6667	7.0835	1.5839	70.3515	76.9818	63.3333	80
	G_{8144}	20	73.5000	7.1308	1.5945	70.1627	76.8373	63.3333	80
	GA	20	60	.0000	.0000	60	60	60	60
	Total	80	69.9583	8.2249	.9196	68.1280	71.7887	60	80

Table 5. Results of the ANOVA test for the "Postoperative" dataset

		Sum of Squares	df	Mean Square	F	Sig.
Fitness training	Between	30.000	2	15.000	2.111	.130
	Within Groups	405.000	57	7.105		
	Total	435.000	59			
Fitness testing	Between	11.481	2	5.741	.122	.886
	Within Groups	2688.333	57	47.164		
	Total	2699.815	59			

Therefore the reduction of the search space by the grammars G_{8344} and G_{8144} does not affect the performance of the BNs generated by EvoBANE.

4 Conclusions

The EvoBANE system has been presented as an evolutionary approach that automatically generates Bayesian networks for solving classification problems.

EvoBANE uses a general-purpose grammar-guided programming core that implements a complete evolutionary mechanism (initialization, selection, crossover and replacement) whose grammatical crossover operator avoids the closure problem.

A CFG generator has been implemented in EvoBANE in order to automate the creation of context-free grammars that generate languages whose sentences codify valid Bayesian network structures that preserve DAG constraints. The CFG generator inputs the specifications of the classification problem to solve and the features of the solutions to be generated, and outputs the CFG that EvoBANE uses to generate and evolve a population of Bayesian networks.

The results show that the Bayesian networks automatically generated by EvoBANE accurately solve classification problems from two different application domains. In these two cases EvoBANE has been able to explore the search space, avoiding local optima and reaching good Bayesian networks without a repair mechanism. The flexibility of the CFG generator vests EvoBANE with the capability to modify the search space in order to find the simplest Bayesian network that solves the problem.

Acknowledgements

This work has been supported by the LIA-Group (http://www.lia.upm.es/) of the Universidad Politécnica de Madrid.

References

1. Wong, M.L., Guo, Y.Y.: Learning bayesian networks from incomplete databases using a novel evolutionary algorithm. Decision Support Systems 45, 368–383 (2008)
2. Darwiche, A.: Bayesian networks. Communications of the ACM 53(12), 80–90 (2010)
3. Niedermayer, D.: An Introduction to Bayesian Networks and Their Contemporary Applications. In: Innovations in Bayesian Networks. SCI, pp. 117–130. Springer, Heidelberg (2008)
4. Chen, X., Anantha, G., Lin, X.: Improving bayesian network structure learning with mutual information-based node ordering in the k2 algorithm. IEEE Transactions on Knowledge and Data Engineering 20(5), 1–13 (2008)
5. Larrañaga, P., Poza, M., Yurramendi, Y., Murga, R.H., Kuijpers, C.M.H.: Structure learning of bayesian networks by genetic algorithms: A performance analysis of control parameters. IEEE Transactions on Pattern Analysis and Machine Intelligence 18(9), 912–926 (1996)
6. Barriére, O., Lutton, E., Wuillemin, P.: Bayesian network structure learning using cooperative coevolution. In: GECCO 2009, Montréal, Canada, July 2009, pp. 755–762 (2009)
7. Alonso-Barba, J.I., delaOssa, L., Puerta, J.M.: Structural learning of bayesian networks by using variable neighbourhood search based on the space of orderings. In: Ninth Conference on Intelligent Systems Design and Applications, pp. 1435–1440 (2009)

8. Font, J.M., Manrique, D., Ríos, J.: Evolutionary construction and adaptation of intelligent systems. Expert Systems with Applications 37, 7711–7720 (2010)
9. Koza, J.: Genetic programming: On the programming of computers by means of natural selection. The MIT Press, Cambridge (1992)
10. Dong, L., Liu, G., Yuan, S., Li, Y., Li, Z.: Classifier learning algorithm based on genetic algorithms. In: ICICIC 2007 Second International Conference on Innovative Computing, Information and Control, September 2007, pp. 126–129 (2007)
11. Larrañaga, P., Kuijpers, C.M.H., Murga, R.H., Yurramendi, Y.: Learning bayesian network structures by searching for the best ordering with genetic algorithms. IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans 26(4), 487–493 (1996)
12. De Stefano, C., Fontanella, F., Scotto di Freca, A., Marcelli, A.: Learning bayesian networks by evolution for classifier combination. In: 10th International Conference on Document Analysis and Recognition, pp. 966–970 (2009)
13. Davidson, C.: Identifying gene regulatory networks using evolutionary algorithms. Journal of Computing Sciences in Colleges 25(5), 231–237 (2010)
14. Wong, M.L., Lee, Y.L., Leung, K.S.: A hybrid approach to learn bayesian networks using evolutionary programming. In: WCCI 2002 Proceedings of the 2002 World Congress on Computational Intelligence, vol. 2, pp. 1314–1319 (2002)
15. Lee, J., Chung, W., Kim, E.: Structure learning of bayesian networks using dual genetic algorithm. IEICE Transactions on Informations and Systems E91-D(1), 32–43 (2008)
16. Palacios-Alonso, M.A., Brizuela, C.A., Sucar, L.E.: Evolutionary learning of dynamic naive bayesian classifiers. Journal on Automated Reasononing 45, 21–37 (2010)
17. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explorations 11(1), 10–18 (2009)
18. Font, J.M., Manrique, D.: Grammar-guided evolutionary automatic system for autonomously building biological oscillators. IEEE Congress on Evolutionary Computation, 1–7 (July 2010)
19. Couchet, J., Manrique, D., Ríos, J., Rodríguez-Patón, A.: Crossover and mutation operators for grammar-guided genetic programming. Soft Computing: A Fusion of Foundations, Methodologies and Applications 11(10), 943–955 (2007)
20. Frank, A., Asuncion, A.: UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences (2010), `http://archive.ics.uci.edu/ml`