Bayesian Network structure Training based on a Game of Learning Automata

S. Gheisari
Department of Computer, Science and Research Branch, Islamic Azad University, Tehran, Iran
S.gheisari@srbiau.ac.ir

M.R. Meybodi
Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran.
mmeybodi@aut.ac.ir

Abstract- Bayesian Network (BN) is a probabilistic graphical model which describes the joint probability distribution over a set of random variables. Finding an optimal network structure based on an available training dataset is one of the most important challenges in the field of BNs. Since the problem of searching the optimal BN structure belongs to the class of NP-hard problems, typically greedy algorithms are used to solve it. In this paper two novel learning automata-based algorithms are proposed to solve the BNs' structure learning problem. In both, there is a learning automaton corresponding with each possible edge to determine the appearance and the direction of that edge in the constructed network; therefore, we have a game of learning automata, at each stage of the proposed algorithms. Two special cases of the game of the learning automata have been discussed, namely, the game with a common payoff and the competitive game. In the former, all the automata in the game receive a unique payoff from the environment, but in the latter, each automaton receives its own payoff. As the algorithms proceed, the learning processes focus on the BN structures with higher scores. The use of learning automata has led to design the algorithms with a guided search scheme, which can avoid getting stuck in local maxima. Experimental results show that the proposed algorithms are capable of finding the optimal structure of BN in an acceptable execution time; and compared with other search-based methods, they outperform them.

Keywords- Bayesian Networks; Game of Automata; Learning Automata; Payoff; Structure Training

1 Introduction

Bayesian networks (BNs) are popular within the AI probability and uncertainty community as a method of reasoning under uncertainty. From an informal perspective, BNs are directed acyclic graphs (DAGs), where the nodes are random variables, and the arcs specify the independence assumptions between these variables. After construction, a BN constitutes an efficient device for performing probabilistic inference [1].

One of the most important challenges of these networks is constructing an optimal network. The optimal network is a network which best reflects the dependence relations in a database of training samples. Searching an optimal structure for BNs is difficult due to the large number of possible DAG structures, given even a small number of nodes to connect. Different algorithms for training BNs from data have been developed; these algorithms can be classified into two categories: constraint-based algorithms, and search-and-score algorithms. In the first category, BNs are constructed by estimating the existence of certain conditional dependencies among nodes [2]; while in the second, heuristic search methods are used to search through the set of feasible structures, and then scoring functions evaluate each structure found [1-6]. Both have their own advantages and disadvantages. Algorithms in the first category can find the optimal network, but on the condition that the data volume is large enough; moreover, due to their conditional independency (CI) tests, they have heavy and impractical computational cost. On the other hand, algorithms in the other category have less time complexity, even in the worst case (when underlying DAG is densely connected), and do not require a large volume of data; however, because of their heuristic natures, they may not find the optimal network [2].

In this paper, two new search-and-score algorithms are proposed for BNs structure training. To search the optimal network, both use a game of learning automata, in which a learning automaton is assigned to each possible undirected edge in the graph. Each automaton determines the appearance and the direction of the corresponding edge. The BNs are built according to the selected actions of every learning automaton. Then, after verifying the networks, scoring functions evaluate the structures using the training samples, and each automaton updates its probability vector based on its received score. Scoring functions evaluate the constructed networks in two different ways; in the first algorithm, the scoring function evaluates whole the network, and then gives a common payoff to all automata. While in the next algorithm, it evaluates each edge in the constructed network and gives different payoffs to each automaton. These steps are run repeatedly in both algorithms until the optimal structures are found. Using the learning automata to perform a guided search leads to find an optimal (or near-optimal) network in lower computation time, and also to improve the score of the constructed network in comparison with rival search methods like hill climbing, genetic search and etc.

The reminder of this paper is organized as follows. In section 2, BNs and structure training, preliminaries are explained. Learning automaton is described briefly in section 3. Section 4 represents the proposed BN structure training algorithms which are based on the game of learning automata. Experimental results obtained by the proposed algorithms are reported in section 5. The paper concludes with a conclusion given in section6.

2 Bayesian Networks

A Bayesian network describes the joint probability distribution over a set of random variables, with defining a series of probability independences and a series of conditional independences [8]. To describe a BN we need to provide two items: topology or structure of the network, and parameters or the conditional probability tables of all variables. Having these two items, one can demonstrate the joint probability distribution among the variables. The joint probability density function (pdf) for n random variables can be written according to Eq. (1).

$$P(X_1, \dots, X_n) = \prod_{i=1}^{n} P(X_i | Parents\ (X_i)) \quad (1)$$

Where $(X_1, \dots, X_n)$ is the set of random variables.

As mentioned before, search-and-score is one category of BN structure training algorithms, which uses heuristic search methods to construct a model. In this category, structure training requires two components: a scoring metric and a search procedure. Scoring metric evaluates the quality of the constructed network; in other words, it determines how much the structure of the constructed BN fits the training samples. Search procedure determines an algorithm to search a network that best fits the training samples according to the received score from the scoring function. In the rest of this section, we briefly describe each of these two components.

2.1 scoring metrics

There are varied metrics proposed for evaluating the structure of BNs such as Bayesian metric, Minimum Description Length metric, and Bayesian Information Criterion, to mention a few. Bayesian metric evaluates a BN by computing a marginal likelihood of the BN by using the given data and inherent uncertainties [9-10]. Minimum Description Length (MDL) metric is based on the assumption that the number of regularities in the data encoded by a model is somehow proportional to the amount of data compression allowed by the model [10]. Finally the Bayesian Information Criterion (BIC), which is used as scoring metric in this work, is a criterion for model selection among a finite set of models, and it is based on the likelihood function. The BIC for a graph, *G,* is given by the following equation [9].

$$BIC = \log_2 P\left(D \middle| G, \hat{\theta}_G\right) - \frac{|n|}{2} \log_2(M) \quad (2)$$

Where D is a set of training samples, $\hat{\theta}_G$ is Maximum Likelihood (ML) estimation for G's parameters, and M is the number of samples in the training dataset. If all variables are multinomial, by considering r_i as a finite set of outputs for x_i, q_i as the number of configurations for x_i's parents, N_{ij} as the number of observations for x_i if j is its parent's configuration, and finally N_{ijk} as the number of observation for $x_i=k$ when its parent's configuration is j, Eq. (2) can be rewritten as below;

$$BIC = \sum_{i=1}^{M} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} log_2 \left(\frac{N_{ijk}}{N_{ij}}\right) - \frac{log_2 N}{2} \sum_{i=1}^{M} q_i (r_i - 1) \quad (3)$$

and in this case BIC converts to a simple counting problem.

2.2 Search procedure

Given a scoring metric, the problem of learning the structure of a Bayesian network belongs to the case of NP-hard problems and there is no polynomial-time algorithm for finding the best network structure corresponding to the most scoring metrics [11]. Usually, a simple greedy algorithm is used to build the network [12-13]. The greedy algorithm adds an edge with the greatest improvement of the current network quality in search step until no more improvement is possible. The initial network structure can be a graph with no edges; furthermore, it can take advantage of using the prior information such as using the best tree computed by the polynomial-time maximum branching algorithm [9] and [12]. Since Bayesian network is an acyclic graph, after each search step, the graph structure must be validated, and all cycles must be removed from the constructed graph [14-16]. To the best of our knowledge, Genetic Algorithms [1], [3], Learning Automata [5], Hill-Climbing [6], Ant Colony optimization [36], and A* search [37] are used in search procedure to build the network.

3 Learning automata

Learning automaton is an adaptive decision making device that tries to learn the optimal action from a finite set of actions in an unknown random environment [17]. This process is done by interacting between the automaton and the random environment. At each time *t,* the automaton selects an action, *α(t)*, according to its action probability distribution, *p(t)*; and it applies the selected action to the random environment. The random environment generates a stochastic reinforcement signal, *β(t),* to the automaton [18]; and the automaton updates the action probability vector using both the reinforcement signal and a learning algorithm. Figure1 represents the interaction between a learning automaton and its environment. Learning automaton, based on its action-set, can be classified into two major classes: finite-action-set learning automata (FALA) and continues-action-set learning automata (CALA). In CALA, actions are Real numbers chosen from a continuous interval [20-22]. For an *r*-action FALA, the action probability distribution is represented by an *r*-dimensional probability vector which is updated by the learning algorithm. Learning automata had been used successfully in many applications such as routing and admission control in computer networks [23-24], solving NP-Complete problems [25-27], and too many other applications [18], [28-31]. For further information about learning automata please refer to [18].

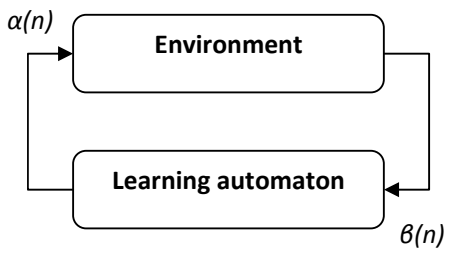

Fig1. Interaction between the learning automaton and the environment

A single automaton is generally sufficient for learning the optimal value of one parameter. However, for multidimensional optimization problems, we need a system consisting of as many automata as there are parameters [17]. Let $A_1, \ldots, A_N$ be the automata involved in an N-player game. Each play of game consists of choosing an action by each learning automaton and then getting the *payoffs* or reinforcement signals from the environment for this choice of actions by the group of learning automata. Let $p_1(k), \ldots, p_N(k)$ be the action probability distributions of N automata; at each instant k, each automaton, Ai, chooses an action $\alpha^i(k)$ independently and at random according to $p_i(k)$, $1 \le i \le N$. This set of N actions is input to the environment, and the environment responds with N random payoffs, which are supplied as reinforcement signals to the corresponding automata. A useful special case of this model is a game with *common payoff*, in which all the automata get the same payoff from the environment (that is, $\beta^i = \beta$). Hence there is only one payoff function. Learning automat in a common payoff game is often referred as a team of learning automata.

4 Proposed algorithms for BN structure training

In this section, we have proposed two novel BN structure training algorithms. They use a game of learning automata to search the optimal structure among all possible structures. First, the learning automata construct a network; then a scoring function evaluates the constructed network, and based on its results, one (in the first proposed algorithm) or several (in the second proposed algorithm) reinforcement signals are produced. Finally, the action probability vectors of all learning automata are updated according to the received signals and a learning algorithm. The training procedures are repeated until termination conditions are satisfied. These algorithms lead to find the optimal structure and decrease the time complexity of constructing a BN.

4.1 BN structure training based on a game of learning automata with a common payoff

Firstly, we have implemented a game with a common payoff. As mentioned before, BN structure training has two components: a scoring function and a search procedure [9-13], and [36]. The proposed algorithm uses BIC, given in Eq. (3), as the scoring function to evaluate the constructed network; and it also uses a team of learning automata, which is a game with a common payoff, to generate the network in a search procedure. In a game with a common payoff, the scoring function gives a single score to the whole network; and the reinforcement signal, which is the response of the environment, is computed according to this score. Then the reinforcement signal is given to all learning automata in the game and they update their probability vectors based on the common signal and a common learning algorithm. The pseudo code of the proposed algorithm is given in Algorithm1.

Algorithm1: Pseudo code of the proposed algorithm with a common payoff

1: Let n be the number of random variables, and m be the number of LAs, i.e. $m = \frac{n(n-1)}{2}$
2: Let t be the iteration counter and initially set to **0** and $\boldsymbol{max_{it}}$ be the maximum number of iterations with $BIC_t < BIC_{t-1}$.
3: **Set** the initial actions probability distribution of each learning automaton to be uniform.
4: **Do** the following steps from 5 to 8, until the current score is not greater than the score of previous $\boldsymbol{max_{it}}$ steps
5: Generate LAs' actions, $(\alpha_1(t), \alpha_2(t), \ldots, \alpha_m(t))$.
6: Build the structure of the BN based on LAs' actions and remove all cycles from the constructed network.
7: Calculate BIC as a scoring metric for current BN.
8: Update the actions probability distribution of each learning automaton based on the reinforcement signal $\beta(t)$.

In what follows we describe the search procedure.

1- BN Construction

Let n be the number of random variables; the maximum number of undirected edges in a graph with n nodes is $m = \frac{1}{2}n \times (n-1)$. We use a team of m learning automata $(A_1, A_2 \ldots A_m)$, with

action set $\{i \to j, j \to i, no\ edge\}\ for\ i = 1, .., n, j = 1, \dots, n\ and\ i \neq j$; each learning automaton is associated to one edge in the graph [5]. Let $\alpha^j(t)$ be the chosen action of automaton *Aj,* at time *t*; it determines either the corresponding edge should not appear in the BN, or it should appear and in this case, it also determines the direction of the edge. The probability vector of automaton *Aj, (for j*=1, 2… *m)* at time *t* is denoted by $p^j(t) = \left(p_0^j(t), p_1^j(t), p_2^j(t)\right)$, where $p_i^j(t)$ is the probability of choosing action α_i (for *i*=0, 1, 2) at time *t*. Initially, there is no prior information about the edges; therefore, we use the uniform probability distribution, i.e. $p_i^j(t) = \frac{1}{3}$ $(for\ i = 0, 1, 2\ and\ j = 1, 2, \dots, m)$. However, the action probability vectors can also be initialized based on the prior information about the variables' dependencies. For example, for each edge the mutual information of the nodes is computed, and then the action probability vector of the corresponding automaton will be initialized based on it (mutual information is explained in section 4.2). Using such prior information speeds up the algorithm and also improves the score of the constructed network; nevertheless, it imposes a new computational overhead to the algorithm. After choosing an action by each learning automaton, a network is built based on their chosen actions.

2- Network verification

The network, which is constructed in previous phase may have cycles. In order to remove cycles, the search procedure uses a modified version of Depth First Search (DFS) algorithm, named colored DFS [32]. It begins with a random start node to navigate the current structure and mark all the back edges, which point from a node to one of its ancestors. After detecting all back edges, the search procedure removes them all. Because of choosing a random start node in the colored DFS, the proposed algorithm has a chance to remove different edges in different iterations, even if the cycles are the same; so the algorithm can find the best structure of the network without biasing to any edge in the network.

3- Compute the reinforcement signal

In the next step, the reinforcement signal should be computed. As mentioned before, here we use a game with a common payoff; and it means that all the automata get the same reinforcement signal from the environment (that is, $\beta^j(t)=\beta(t)$). The BIC, given in Eq. (3) in section 2.1, is used to score the whole network, and then the common reinforcement signal at *t*, $\beta(t)$, is computed using Eq. (4).

$$\beta(t) = Max\left(\frac{BIC_t - BIC_{t-1}}{BIC_{max} - BIC_{min}}, 0\right) \quad (4)$$

Where, BIC_t and BIC_{t-1} are the scores of the constructed networks in the current iteration, *t*, and the previous iteration, *t-1*, respectively; BIC_{max} is the maximum score, and BIC_{min} is the minimum score, which are gained up to the current iteration, *t*.

4- Update the action probability vector

At each stage, *t*, the learning algorithm updates the action probability distribution of every automata using following equation,

$$p^j(t+1) = p^j(t) + \lambda\beta(t)\left(e_{\alpha_j} - p^j(t)\right)\ \ j = 1, \dots, m \quad (5)$$

where $0 < \lambda < 1$ is a constant parameter, and e_{α_j} is a unit vector of appropriate dimension (here the dimension is three that is the number of actions of each learning automaton) with α_jth component unity. $\beta(t) \in [0,1]$, and this algorithm is known as Linear Reward-Inaction (L_{R-I}) algorithm.

5-Termination

Step 1, 2, 3 and 4 are repeated until the stage number exceeds a pre-specified threshold Th_{max}, or the score of the constructed BN becomes greater than a certain pre-defined score BIC_{opt}, or for a pre-defined iterations max_{it}, the score of a constructed BN does not change. Here the latest state is used as the termination criteria.

However maximizing the expected value of the common payoff β, which depends on the BIC value, is the main goal of the team of learning automata; reaching a *Nash equilibrium* is another important goal in a game. For good discussion on the rationality of Nash equilibrium, please refer to [33]. To explain more, let $S^j = \{\alpha_1^j, \alpha_2^j, \alpha_3^j\}$ is the action set of automaton A_j for $1 \leq j \leq m$. Define $S=\prod_{j=1}^{m} S_j$; for any $\boldsymbol{a} = (a_1, \ldots, a_m) \in S$, we define its neighborhood in S as $N(\boldsymbol{a}) = \{x \in S | \exists i, s.t.\ x_j = a_j, \forall j \neq i,\ and\ x_i \neq a_i\}$. Thus, the neighbors of any m-tuple of actions are the set of all action choices that differ only in one action. It is easy to see that a specific m-tuple of actions $\boldsymbol{a} \in S$ is a Nash equilibrium for this game with a common payoff if and only if $\beta(\boldsymbol{a}) \geq \beta(x)$, $\forall x \in N(a)$. The corresponding β which is the highest is called β_{op}. It has been proven that the automata team would converge to one of the Nash equilibrium, if all the automata use L_{R-I} (with sufficiently small value for the step size parameter) [33].

4.2 Structure training based on a competitive game of learning automata

In the rest of this section, we will introduce a competitive game of learning automata for BN structure training. In a competitive game, each learning automaton receives its reinforcement signal, $\beta^j(t)$, from the environment; it means that the scoring function should give different scores. Then, all automata update their probability vectors according to their own reinforcement signals and a common learning algorithm. The pseudo code of the proposed algorithm is given in Algorithm 2.

Algorithm2: Pseudo code of the proposed algorithm with different payoffs

1: Let n be the number of random variables, and m be the number of LAs, i.e. $m = \frac{n(n-1)}{2}$.
2: Let t be the iteration counter and initially set to **0** and $\boldsymbol{It._{max}}$ be the maximum number of iterations for running the algorithm.
3: **Set** the initial actions probability distribution of each learning automaton to be uniform.
4: **Do** the following steps from 5 to 8 for $\boldsymbol{It._{max}}$
5: Generate LAs' actions, $(\alpha_1(t), \alpha_2(t), \ldots, \alpha_m(t))$.
6: Build the structure of the BN based on LAs' actions and remove all cycles from the constructed network.
7: Calculate $I_j(t)$ for each edge in the BN as the mutual information of its two nodes.
8: Update the actions probability distribution of the corresponding learning automata based on the reinforcement signal $\beta_j(t)$.

The search procedure of this algorithm is the same as the previous one. At each stage t, automaton A_j, chooses an action $\alpha^j(t)$, independently and at random according to $p^j(t)$. The chosen set of m actions is used to build a BN structure. Like before, the selected action of each automaton determines either the corresponding edge should not appear in the BN, or it should appear and in this case, it also determines the direction of the edge. After cycles detecting and removing by a colored DFS algorithm [32], the constructed network should be evaluated. Here scoring function produces m scores (for m learning automata); and these scores are individually given to the automata, as their payoffs or reinforcement signals, $\beta_j(t)$. In order to generate the reinforcement signal of each learning automaton, which is assigned to an edge, mutual information of the nodes of the edge, at stage t is computed using Eq. (6), and the training datasets.

$$I_t(X_l, X_k) = \sum_{x_l, x_k} P(x_l, x_k) log \frac{P(x_l, x_k)}{P(x_l)P(x_k)} \quad (6)$$

Where X_l and X_k are the corresponding nodes. The mutual information measures the amount of information flows between two nodes. It is also used in some of the BN constructing algorithms that belong to the first category of the researches (As mentioned in section 1, the first category of the researches is based on analyzing dependency relationships among nodes [2].). The reinforcement signal of the corresponding automaton, A_j at time t, is **one** either $I_t(X_l, X_k) > \varepsilon$ and the chosen action is $l \rightarrow k$, or $I_t(X_l, X_k) \leq \varepsilon$ and the chosen action is $no\ edge$; otherwise, $\beta_j(t)$ is **zero**; where ε is a certain threshold value.

After computing the $\beta_j(t)$, the learning algorithm updates the action probability distribution of each automaton using the following equation,

$$p^j(t+1) = p^j(t) + \lambda\beta_j(t)\left(e_{\alpha_j} - p^j(t)\right) \quad j = 1, \ldots, m' \tag{7}$$

Where $0 < \lambda < 1$ is a constant parameter and e_{α_j} is a unit vector of appropriate dimension, (here the dimension is three that is the number of actions of each learning automaton) with α_jth component unity. In this section, we assumed that $\beta_j(t) \in \{0,1\}$, but the learning algorithm is still L_{R-I}. All steps are repeated for a certain number of iterations, called $It._{max}$.

The same as the game with a common payoff, it is expected that the algorithm reaches a Nash equilibrium; despite the fact that, each automaton has its own payoff and tries to improve it. The m-tuple of actions $(a_1, \ldots, a_m)$ is called a Nash equilibrium of this game if for each j, $1 \leq j \leq m$,

$$d^j(a_1, \ldots, a_{j-1}, x, a_{j+1}, \ldots, a_m) \leq d^j(a_1, \ldots, a_m) \forall x \in S_i \tag{8}$$

Where d^j is the payoff function for automaton A_j ($d^j(x_1, \ldots, x_m) = E[\beta_j(t) | \alpha_j(t) = x_j, 1 \leq j \leq m]$), and S_i is the action set of A_j.

It is proven in [33] that, if the learning algorithms of all the learning automata are L_{R-I}, this game would converge to a Nash equilibrium.

5 Experimental Results

In this section the proposed LA-based algorithms are evaluated. To do this, we have designed two categories of experiments. At first, a well-known BN named ALARM [34], is chosen; and two LA-based algorithms are employed to construct the ALARM network using its available datasets. Then constructed networks are evaluated and the proposed algorithms are compared with each other and with other BN structure training algorithms. Second, since classification is one of the important applications of BNs, for 25 chosen datasets the classification accuracies of the proposed algorithms are compared against other algorithms.

5.1 Results on the ALARM Network

To evaluate the performance of the algorithms, experiments have been designed in following steps the same as [1]:

Step1: Determine a well-known BN (structure and conditional probabilities) and simulate it, obtaining its database of cases D, which must reflect the conditional independence relations between the variables.

Step2: Using the LA-based algorithms, try to obtain the networks $BN_{LA\text{-}CP}$ and $BN_{LA\text{-}DP}$, which maximize the probabilities $P(D| BN_{LA\text{-}CP})$ and $P(D| BN_{LA\text{-}DP})$.

Step3: Evaluate the $BN_{LA\text{-}CP}$ and the $BN_{LA\text{-}DP}$, and then compare them with other BNs' structures, which are constructed by other algorithms (all algorithms are described in section 5.1).

The BN used in the experiments is the ALARM network. ALARM network is a medical diagnostic alarm message system for patient monitoring; it contains 37 nodes and 46 arcs (see figure 2).

Researchers in this field have used datasets which were generated from three versions of ALARM network with the same structure but different probability distributions [2]. We use 5000 first cases from the database that was generated by Edward Herskovits [35].

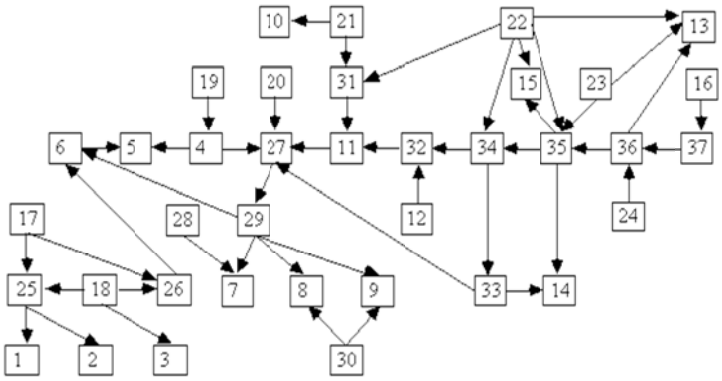

Fig2. The ALARM network

5.1.1 Sensitivity analysis

First, we start with a sensitivity analysis on the learning parameter λ, in order to study the effect of λ on the performance of the LA-CP and the LA-DP and also to find the best value of it. To do this, the database with 5000 cases is considered. Figure 3 represents the results. It can be seen that the best value of the parameter λ for LA-CP is between 0.4 and 0.5, and for LA-DP is between 0.3 and 0.4.

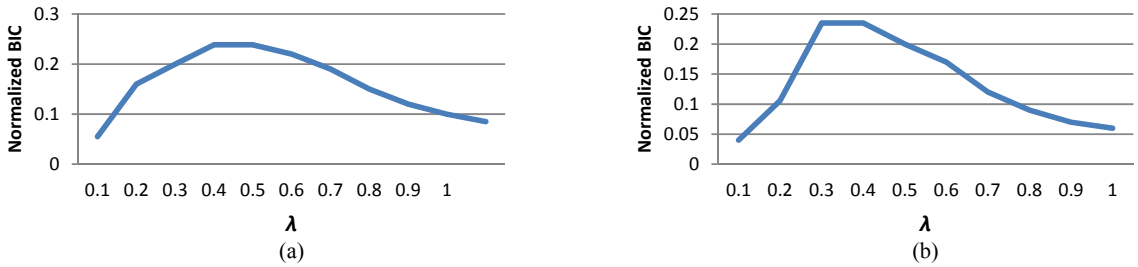

Fig3. Analysis the effect of parameter λ for (a) LA-CP and (b) LA-DP

5.1.2 Construct the ALARM network

In next experiments, the proposed algorithms are employed to construct the ALARM network using the available training samples. Both algorithms construct the networks $BN_{LA\text{-}CP}$ (based on the game of LAs with a common payoff) and $BN_{LA\text{-}DP}$ (based on the competitive game of LAs with different payoffs) which are identical to the ALARM, except that two arcs {21-31 and 12-32} are missed. A subsequent analysis has revealed that missing arcs are not supported by the 5000 cases, and their nodes are actually independent in the employed database. It is similar to the result of [2] which used 10000 cases and nodes ordering.

Then, different subsets of the dataset are considered; the subsets are made by the first 500, 1500, 3000, and 5000 cases. For each database of cases, the following values are measured;

I. BIC as a scoring metric,
II. Hamming distance between the constructed networks, and the original ALARM network,
III. Computation time.

BIC is measured by Eq. (3). The interpretation of BIC is: the higher this parameter, the better the network. Hamming Distance directly compares the structure of the constructed network with the structure of the original network. We have defined the Hamming Distance between two DAGs as the

number of the following operators required to make the DAGs match: add, remove, or reverse a directed edge. The lower Hamming distance indicates the more similar network with the ALARM network. LA-CP and LA-DP are also compared based on their computation time. They are implemented in .Net framework in a PC which has a single CPU of Intel(R) Core™ 2 Duo 3.33GHz and a 1GB memory. In order to measure the computation time, each algorithm is executed with no prior limitation in the number of iterations, until no improvement in BIC is observed.

Table 1 represents the average results. For all datasets with 500, 1500, 3000, and 5000 cases, the algorithms are implemented 10 times independently. As the results indicate, using databases of 3000 and 5000 cases, both LA-based algorithms perform well and construct acceptable networks (networks with low Hamming distance and high BIC score); however, for databases of 500 and 1500 cases, the performance of LA-DP decreases, compared with LA-CP. Such results were predictable because in LA-DP scoring metric is the mutual information between the corresponding nodes of each edge, and in this situation, in order to have a reliable scoring metric, the volume of data should be large enough. The computation time is also measured for both algorithms, and LA-CP has consumed less time in comparison with LA-DP.

Table1. Average results of proposed algorithms after 10 runs

Number of cases Parameters\ Networks	500		1500		3000		5000	
	$BN_{LA\text{-}CP}$	$BN_{LA\text{-}DP}$	$BN_{LA\text{-}CP}$	$BN_{LA\text{-}DP}$	$BN_{LA\text{-}CP}$	$BN_{LA\text{-}DP}$	$BN_{LA\text{-}CP}$	$BN_{LA\text{-}DP}$
Normalized BIC	0.039822	0.01235	0.15461	0.068914	0.238	0.15055	0.23853	0.235064
Hamming Distance	17.7	35.0	4.1	19.0	2.0	7.3	2.0	2.0
Computation time	71.30	141.91	81.19	152.54	96.52	166.94	116.14	189.62

5.1.3 Compare with other structure training algorithms

At the end of this section, the proposed LA-based algorithms are compared against other BN structure training algorithms. Below all implemented algorithms are described:

- Proposed algorithm based on a game of learning automata with a common payoff, called LA-CP.
- Proposed algorithm based on a competitive game of learning automata with different payoffs, called LA-DP.
- 3-phase network construction algorithm which is based on computing the mutual information of attributes pairs.
- Max-Min Hill climbing algorithm [6].
- Genetic algorithm which finds the best order of variables and then starts the search process [3].
- Ant-colony optimization [36] called ACO.

Many other BN structure training algorithms exist; A* search-based algorithm with a shortest path perspective [37], and multiple exact algorithms [38-40] are some examples. However, it is impossible and unnecessary to compare the proposed algorithms with all the structure training algorithms; so we have selected more comprehensive algorithms, which also stand in the same class with ours.

Table2. Compare the proposed algorithms with other algorithm

	Normalized BIC	Hamming Distance	Computation time
LA-DP	0.23506	2.0	189.62
LA-CP	0.23853	2.0	116.14
3-phase	0.06232	9.5	671.35
Max-Min hill climbing	0.20781	2.0	308.02
Genetic algorithm-based	0.14404	2.6	284.15
Ant- colony optimization	0.14402	2.6	259.12

Datasets of 5000 cases are considered, and the performance metrics are defined like the previous experiments. Algorithms are implemented with no prior limitation in time and until no improvement in the score is observed. Table2 shows the results after 10 independent runs.

5.2 Evaluate the predictive ability of the proposed algorithms in classification

A constructed BN is an efficient device to perform probabilistic inference; therefore, predictive ability in different applications is one of the important issues in the field of BNs. Classification is one of the important applications of BNs which is used in varied fields such as recommender systems for estimating users' ratings based on their implicit preferences, bank direct marketing for predicting clients' willingness of deposit subscription, disease diagnosis for assessing patients' breast cancer risk, simultaneous fault diagnosis problem, and classification problems with continuous attributes [41-43]. In this section first we briefly explain about BNs classifiers in section 5.2.1, and then choosing datasets of different applications, we evaluate the classification accuracy and the classification time of different algorithms in sections 5.2.2 and 5.2.3.

5.2.1 BN classifiers

Suppose that each training sample is a vector of attributes $(X_1, X_2 \ldots X_{v-1}, C)$. The goal of classification is predicting the right value of class variable $c=x_v$ having $(x_1, x_2 \ldots x_{v-1})$. If the performance measure is the percentage of correct predictions on the test samples (classification accuracy), correct prediction for $(x_1, x_2 \ldots x_{v-1})$ is the class that maximizes $P(c|x_1, x_2 \ldots x_{v-1})$. If there is a BN over $(x_1, x_2 \ldots x_{v-1}, C)$, we can compute these probabilities by inference on it. After the structure of a BN is specified, estimating the parameters so that the network can provide the best prediction for the value of the class variable in the test samples is important; however, it is out of the scope of this study, and we simply use Maximum Likelihood (ML) to estimate the parameters' values.

Table3. Datasets and their samples

Dataset	Number Of classes	Number Of Attributes	Number Of samples	Dataset	Number Of classes	Number Of attributes	Number Of samples
australian	2	14	690	Iris	3	4	150
breast	2	10	683	letter	26	16	15000
chess	2	36	2130	lymphography	4	18	148
cleve	2	13	296	mofn-3-7-10	2	10	300
corral	2	6	128	pima	2	8	768
crx	2	15	653	shuttle-small	7	9	3866
diabetes	2	8	768	vote	2	16	435
flare	2	10	1066	satimage	6	36	4435
german	2	20	1000	segment	7	19	1540
glass	7	9	214	soybean-large	19	35	562
glass2	2	9	163	vehicle	4	18	846
heart	2	13	270	waveform-21	3	21	300
hepatitis	2	19	80				

5.2.2 Comparison the classification accuracy of different classifiers

In order to evaluate and compare the classification accuracy of the proposed algorithms, 25 datasets are used, which consist of 21 datasets from UCI [44] and others from [45]. Table3 shows a brief description of these datasets.

All implemented classifiers are described as follows.

- LA-CP and LA-DP classifiers,
- Naïve Bayes classifier,
- TAN classifier,
- 3-phase network construction algorithm as a classifier,
- Hill climbing-based classifier,
- And Genetic algorithm-based classifier [3].

The ACO-based algorithm is eliminated in the following experiments because its performance is very similar to the performance of the GA-based algorithm. We also compare the performance of the proposed algorithm with two simple and well-known classifiers, named Naïve Bayes and TAN.

Furthermore, in order to construct more efficient networks in classification, another scoring function is used, which is proposed in [46] and called classification rate,

$$CR = \frac{1}{|D|}\sum_{m\in D}^{|D|} \delta(B_D(x_{1:N}^m), c^m) \quad (9)$$

where, $|D|$ is the number of training samples. The equation simply represents the rate of samples that are classified correctly by the network. And $\delta(B_D(x_{1:N}^m), c^m) = 1$ if BN classifier $B_D(x_{1:N}^m)$ which is trained with D, predicts the right value of class variable c^m having attributes $x_{1:N}^m$.

Table4 represents the classification error rate of implemented algorithms for different datasets; the best results are highlighted. Experiments are repeated 10 times with a fixed execution time, and the results are averaged. Average error rates (which are **0.133864** for LA-CP, 0.146492 for LA-DP, 0.18094 for NB, 0.1586 for TAN, 0.159376 for 3-phase, 0.171016 for HC, and 0.146768 for GA) indicate that LA-CP and LA-CP have performed better than other classifiers, on average. Moreover, for 14 datasets LA-CP, and for five datasets LA-DP, have shown the superior results. For remaining datasets, HC on two datasets, TAN on two datasets, 3-phase on one dataset, and BN on one dataset, have shown the best results. However, for all 11 datasets the differences between the best results and the results achieved by LA-CP are negligible; and the slight differences between them are justifiable by considering the stochastic nature of the algorithms.

In addition, figure4 represents the scatter charts of comparing the algorithms. Firstly, we compare LA-CP with LA-DP, and then, since the LA-CP shows much better performance, we compare it with other algorithms. In Figure4, points above the line y=x, show wherever LA-CP has better performance in comparison with the other algorithm.. Figure5 shows bar chart, which compares the average error rate of all algorithms on different datasets. It is clear that LA-CP is indicating better results.

5.2.3 Comparison the classification time of different classifiers

Finally, the classification time of different algorithms are measured. All algorithms are implemented and run in .Net framework in a PC which has a single CPU of Intel(R) Core™ 2 Duo 3.33GHz and a 1GB memory. To measure the execution time, algorithms are run with no prior limitation in time until no improvement is observed. Table5 shows the results after 10 independent runs for seven chosen dataset. Chosen datasets are: Letter, Chess, German, Hepatitis, Breast, Glass2, and Iris. The best results are highlighted. Results again show that LA-CP requires less time for classification.

Table4. Average error rate for different datasets

algorithms / datasets	LA-CP	LA-DP	NB	TAN	3-phase	HC	GA
Australian	0.1341	**0.1149**	0.1489	0.1751	0.1296	0.1391	0.1205
Breast	**0.0210**	0.0411	0.0245	0.0351	0.0425	0.0668	0.0522
Chess	**0.0312**	0.0469	0.1266	0.076	0.0422	0.0966	0.0459
Cleve	0.1894	0.1863	**0.1791**	0.2164	0.1997	0.182	0.1826
Corral	0.0041	0.0197	0.1277	0.0143	0.0119	**0**	0.0177
Crx	0.1392	0.1401	0.1505	0.1631	0.158	**0.137**	0.146
Diabetes	**0.2168**	0.268	0.2571	0.2384	0.2666	0.255	0.2743
Flare	0.1823	**0.174**	0.2024	0.178	0.1804	0.181	0.1762
German	**0.2412**	0.2551	0.2458	0.2609	0.2643	0.3085	0.2573
Glass	**0.3012**	0.3992	0.4412	0.4578	0.4173	0.4412	0.4061
Glass2	**0.1634**	0.2413	0.2236	0.2249	0.2691	0.2329	0.1758
Heart	**0.1526**	0.1751	0.155	0.1847	0.1874	0.2221	0.1789
Hepatitis	**0.09382**	0.1728	0.193	0.1302	0.1618	0.1967	0.1529
Iris	**0.0401**	0.0411	0.0699	0.0763	0.042	0.0414	0.063
Letter	0.23	**0.1061**	0.3068	0.1752	0.183	0.1896	0.1256
Lymphography	**0.1396**	0.1554	0.1662	0.1784	0.1635	0.2247	0.1542
Mofn-3-7-10	**0.0800**	0.0866	0.1328	0.085	0.0859	0.0859	0.0928
Pima	**0.1375**	0.1823	0.2571	0.2384	0.2666	0.255	0.182
Satimage	0.1771	0.1867	0.1915	**0.1395**	0.1745	0.184	0.1743
Segment	0.0944	0.0662	0.1221	0.0675	**0.0571**	0.0831	0.0592
Shuttle-small	0.0090	**0.0041**	0.014	0.0093	0.0047	0.0145	0.0066
Soybean-large	**0.0629**	0.0786	0.0852	0.0644	0.0754	0.0922	0.0765
Vehicle	0.2836	0.2987	0.3892	**0.2718**	0.2922	0.3451	0.2994
Vote	**0.0319**	0.0374	0.0991	0.0509	0.0417	0.0467	0.0424
Waveform-21	0.1892	**0.1846**	0.2142	0.2534	0.267	0.2543	0.2068
Average	**0.133864**	0.146492	0.18094	0.1586	0.159376	0.171016	0.146768

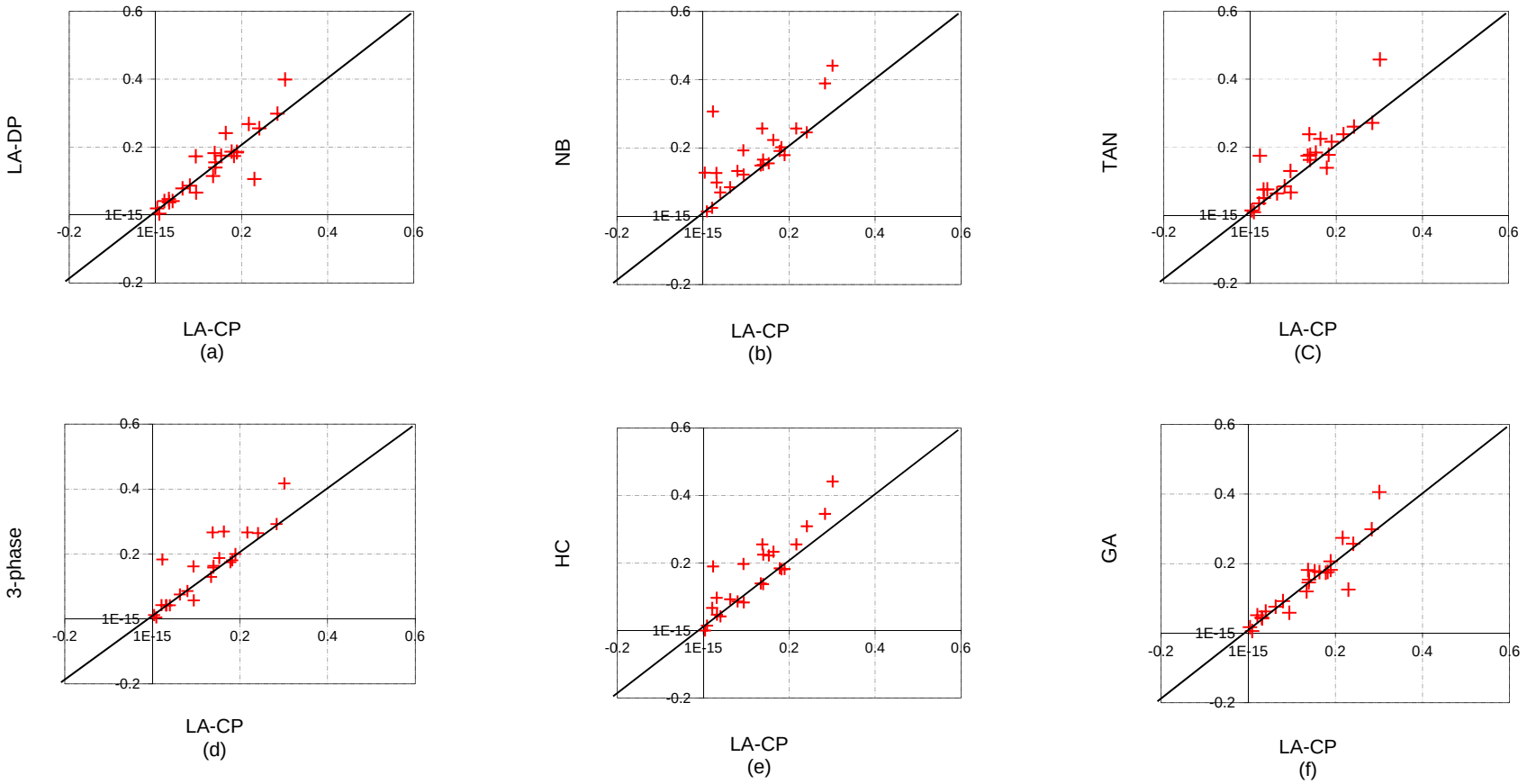

Fig4. Scatter chart to compare the classification error of LA-CP with (a) LA-DP, (b) Naïve Bayes, (c) TAN, (d) 3-phase, (e) Hill Climbing-based, and (f) Genetic algorithm-based classifiers; points above y=x show better performance of the LA-CP.

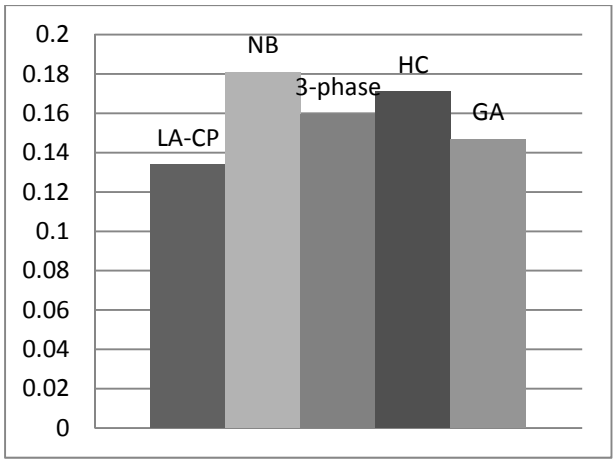

Fig5. Average classification error rate

Table5. Comparing the execution time in classification

algorithms / datasets	LA-DP	LA-CP	NB	TAN	3-phase	HC	GA
Letter	386.49	**136.09**	676.42	458.59	898.81	608.64	458.59
Chess	195.93	**71.93**	357.36	288.45	570.22	309.88	288.45
German	88.12	**33.12**	148.42	95.45	164.80	106.32	95.45
Hepatitis	64.99	**21.99**	52.54	88.43	93.01	89.34	88.43
Breast	21.42	**13.42**	82.73	28.43	241.09	35.13	28.43
Glass2	18.14	**7.14**	28.45	24.39	49.41	26.14	24.39
Iris	8.42	**4.32**	19.90	14.54	29.29	16.12	14.54

6. Conclusion

In this paper two novel learning automata-based algorithms are proposed, for structure training in BNs. They try to perform a guided search through the space of the possible network structures. In order to construct a BN, the first algorithm uses a team of learning automata in a game with a common payoff, named LA-CP, and the second, named LA-DP, uses a competitive game with different payoffs. In both, there is one learning automaton for each possible edge in the undirected related graph, and each learning automaton tries to learn the existence or not existence of the corresponding edge. In each stage, one BN is constructed based on the selected actions of the learning automata; then each learning automaton update its action probability vector based on the evaluation of the constructed network in LA-CP, and the evaluation of each edge in LA-DP. Some experiments have been developed to evaluate the performances of the algorithms. The first class of experiments is designed to compare two proposed algorithms with each other. Results show that LA-CP has superior performance; and the performance of LA-DP depends on the volume of the training dataset because it uses the mutual information as a scoring metric to evaluate each edge separately. Next, in order to compare the proposed algorithms with other BN structure training algorithms, some algorithms, which sit in the same class with ours, are selected. Comparison is done for both structure training, and classification accuracy, which is an important application of the BN.

Experimental results for structure training indicate the better performance of the proposed algorithms compared with others. Measures are the quality of the constructed networks, which is evaluated by using BIC and Hamming distance, and also the execution time. Furthermore, to examine the classification accuracy of the proposed algorithms, classification error of different BNs on 25 datasets from [44-45] are measured. Experimental results show that the LA-CP performs better than other classifiers, on average.

Structure learning of BNs and Bayesian classifiers are still active fields of research; writers are eager to continue their study in this field and improve some features of LA-based algorithms; for example, speed up the convergence rate, reduce the number of learning automata, and etc. Moreover, other class of BN, which is known as Dynamic BN, is an interesting research topic; and structure learning of DBNs based on learning automata, will be the subject of the writers' next investigation.

References

[1] P. Larranaga, M. Poza, Y. Yarramendi, R.H. Murga, and C.M.H. Kuijpers, "*Learning of Bayesian Networks by Genetic Algorithms: A Performance Analysis of Control Parameters*", IEEE Transactions, Pattern Analysis and Machine Intelligence, Vol. 18, pp. 912-926, 1996.

[2] J. Cheng, D.A. Bell, and W. Liu, "*An Algorithm for Bayesian Belief Network Construction from Data*", In Proceedings of AI & STAT'97, pp. 83-90, 1997.

[3] P. Larranaga, C.M.H. Kuijpers, R.H. Murga, and Y. Yurramendi, "*Learning Bayesian Network Structures by Searching for the Best Ordering with Genetic Algorithms*", IEEE Transactions on, Systems, Man and Cybernetics, Vol. 26, pp. 487-493, 1996.

[4] J.W. Myers, K.B. Laskey, and T.S. Levitt, "*Learning Bayesian Networks from Incomplete Data with Stochastic Search Algorithms*", UAI Conference, pp.476-485, 1999.

[5] N. Rezvani, and M.R. Meybodi, *"A Learning Automata-based Technique for Training Bayesian Networks"*, Proceeding of international conference CACTE, pp. 201-212, 2009.

[6] I. Tsamardinos, L.E. Brown, and C.F. Aliferis, *"The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm"*, Mach Learn, Vol. 65, pp. 31-78, 2006.

[7] D. Heckerman, "*Learning with Bayesian Networks*", ICML'95, 1995.

[8] K.P. Murphy, "*An Introduction to Graphical Models*", Technical Report, Intel Research Technical Report, 2001.

[9] Lam, W. & Bacchus, F., "*Learning Bayesian Belief Networks: An Approach Based on the MDL Principle*", Elsevier, Computational Intelligence, Vol. 10, pp. 269-293, 1994.

[10] M. Gallagher, I. Wood, J. Keith, *"Bayesian Inference in Estimation of Distribution Algorithms"*, In Proceeding of the IEEE Congress on Evolutionary Computation, pp. 127-133, 2007.

[11] D.M. Chickering, D. Geiger, D. Heckerman, *"Learning Bayesian Network is NP-hard"*, Technical Report MSR-TR-94-14, 1994.

[12] D. Heckerman, D. Geiger, D.M. Chickering, *"A tutorial on Learning with Bayesian Networks"*, Innovations in Bayesian Networks, Chapter 3, Springer, Berlin, pp. 33-82, 2008.

[13] C.W. Ahn, R.S. Ramakrishna, *" On the Scalability of Real-coded Bayesian Optimization Algorithm"*, IEEE Transactions, Evol. Comut. Vol. 12, pp. 307-322, 2008.

[14] C.W. Ahn, R.S. Ramakrishna, D.E. Goldberg, *"Real-coded Bayesian Optimization Algorithm: Bringing the Strength of BOA into the Continuous World"*, Lecture Notes in Computer Science, Vol. 3102, pp. 840-851, 2004.

[15] D. Heckerman, D. Geiger, D.M. Chickering, *"Learning Bayesian Networks: The Combination of Knowledge and Statistical Data"*, Technical Report MSR-TR-94-09, 1995.

[16] C.W. Ahn, *"Advances in Evolutionary Algorithms: Theory, Design and Practice"*, Studies in Computational Intelligence, Springer, 2006.

[17] M.A.L. Thathachar, P.S. Sastry, *"Varieties of Learning Automata: an Overview"*, IEEE Transactions on System, Man, and Cybernetics,. Vol. 32, pp. 711-722, 2002.

[18] K.S. Narendra,and M.A.L. Thathachar, , "*Learning Automata: An Introduction*", Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1989.

[19] K. Najim, and A.S. Poznyak, "*Learning Automata: Theory and Applications*" Pergamon Press, Inc. Elmsford, NY, USA, 1994.

[20] H. Beigy, and M.R. Meybodi, *"A New Continuous Action-set Learning Automata for Function Optimization"*, J. Franklin Inst. Vol. 343, pp. 27, 2006.

[21] G. Santharam, P.S. Sastry, and M.A.L. Thathachar, *"Continuous Action Set Learning Automata for Stochastic Optimization"*, J. Franklin Inst. Vol. 331B, pp. 607-628, 1994.

[22] B.J. Oommen, and T.D. Roberts, *"Continuous Learning Automata Solutions to the Capacity Assignment Problem"*, IEEE Transactions on Computer, Vol. 49, pp. 608-620, 2000.

[23] M.S. Obaidat, G.I. Papadimitriou, A.S. Pomportsis, and H.S. Laskaridis, "*Learning automata-based bus arbitration for shared-medium ATM switches*", IEEE Transactions on System, Man, and Cybernetics, Vol. 32, PP. 815–820, 2002.

[24] G.I. Papadimitriou, M.S. Obaidat, and A.S. Pomportsis, "*On the use of learning automata in the control of broad-cast networks: a methodology*", IEEE Transactions on System, Man, and Cybernetics, Vol. 32, PP. 781–790, 2002.

[25] H. Beigy, and M.R. Meybodi, "*Cellular learning automata based dynamic channel assignment algorithms*", Int. J. Comput. Intell. Appl. Vol. 8(3), PP. 287–314, 2009.

[26] H. Beigy, and M.R. Meybodi, "*Utilizing distributed learning automata to solve stochastic shortest path problems*", Int. J. Uncertain Fuzz Knowl. Based Syst. Vol. 14, PP. 591–615, 2006.

[27] O.C. Granmo, B.J. Oommen, S.A. Myrer, and M.G. Olsen, *"Learning automata-based solutions to the nonlinear fractional knapsack problem with applications to optimal resource allocation*", IEEE Transactions on System, Man, and Cybernetics, Vol. 37, PP. 166–175, 2007.

[28] H. Beigy, and M.R. Meybodi, *"Adaptive limited fractional guard channel algorithms a learning automata approach"*, Int. J. Uncertain. Fuzziness Knowl. Based Syst. Vol. 17(6), PP. 881–913, 2009.

[29] H. Beigy, and M.R. Meybodi, *"A learning automata-based algorithm for determination of the number of hidden units for three layer neural networks"*, Int. J. Syst. Sci. Vol. 40, PP. 101–118, 2009.

[30] P.S. Sastry, G.D. Nagendra, and N. Manwani, *"A team of continuous-action learning automata for noisetolerant learning of half-spaces",* IEEE Transactions on System, Man, and Cybernetics, Vol. 40, PP. 19–28, 2010.

[31] B.J. Oommen, and M.K. Hashem, *"Modeling a student classroom interaction in a tutorial-like system using learning automata*", IEEE Transactions on System, Man, and Cybernetics, Vol. 40, PP. 29–42, 2010.

[32] Amir Kamil, *"Graph Algorithms"*, CS61B, Spring , UC Berkeley,2003

[33] P.S. Sastry, V.V. Phansalkar, and M.A.L. Thathachar, *"Decentralized Learning of Nash Equilibria in Multi-Person Stochastic Games with Incomplete Information"*, IEEE Transactions on System, Man, and Cybernetics, Vol. 24, pp. 769-777, 1994.

[34] I.A. Beinlich, H.J. Suermondt, R.M. Chavez, and G.F. Cooper, *"The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks"*, Proceedings of Second European Conference on Artificial Intelligence, pp. 247-256, 1989.

[35] E.H. Herskovits, *"computer Based Probabilistic Network Construction"*, doctoral dissertation, Medical Information Sciences, Stanford Univ. 1991.

[36] De Campos, Luis M., Juan M. Fernandez-Luna, José A. Gámez, and José M. Puerta. "Ant colony optimization for learning Bayesian networks." *International Journal of Approximate Reasoning* 31, no. 3 (2002): 291-311.

[37] Yuan, Changhe, Brandon Malone, and Xiaojian Wu. "Learning optimal Bayesian networks using A* search." In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 3, p. 2186. 2011.

[38] Singh, Ajit P., and Andrew W. Moore. "Finding optimal Bayesian networks by dynamic programming." (2005).

[39] Cussens, James, and Mark Bartlett. "Advances in Bayesian network learning using integer programming." *arXiv preprint arXiv:1309.6825* (2013).

[40] Jaakkola, Tommi, David Sontag, Amir Globerson, and Marina Meila. "Learning Bayesian network structure using LP relaxations." In *International Conference on Artificial Intelligence and Statistics*, pp. 358-365. 2010.

[41] X. Z. Wang, Y. L. He, and D. D. Wang. Non-Naive Bayesian Classifiers for Classification Problems with Continuous Attributes. IEEE Transactions on Cybernetics, 2014, 44(1): 21-39.

[42] Yulin He, Ran Wang, Sam Kwong, Xizhao Wang, Bayesian classifiers based on probability density estimation and their applications to simultaneous fault diagnosis, Information Sciences, 2014, 259: 252-268.

[43] Feng, Guang, Jia-Dong Zhang, and Stephen Shaoyi Liao. "A novel method for combining Bayesian networks, theoretical analysis, and its applications." *Pattern Recognition* 47, no. 5 (2014): 2057-2069.

[44]P.M. Murphy, and D. W. Aha, "*UCI Repository of Machine Learning Databases*", Available: http://www.ics.uci.edu/~mlearn/MLRepository.html, 1995.

[45] R. Kohavi, and G. John, "*Wrappers for Feature Subset Selection*", Elsevier Science, Artificial Intelligence, Vol. 97, pp. 273-324, 1997.

[46] Pernkopf, Franz. "Bayesian network classifiers versus selective k-NN classifier." *Pattern Recognition* 38, no. 1 (2005): 1-10.

Appendix A. Experiments Implementation

In order to evaluate the performance of proposed algorithms and compare them with other algorithms, required programs are implemented with C# on .Net Framework. We have implemented two main elements called DataModel and BayesModel. In DataModel preprocess of datasets is done and an object oriented model is used to maintain the datasets. BayesModel constructs a BN based on a given dataset and then evaluates it. Figure 6 and figure 7 represent their class diagram.

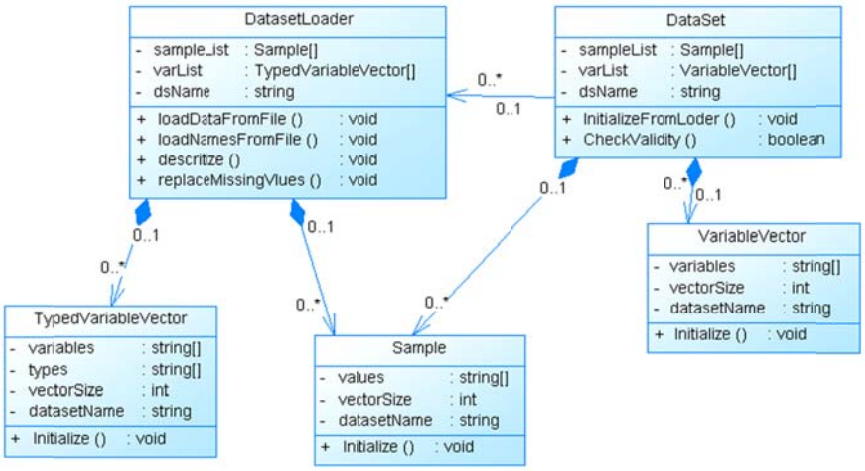

Fig6. DataModel class diagram

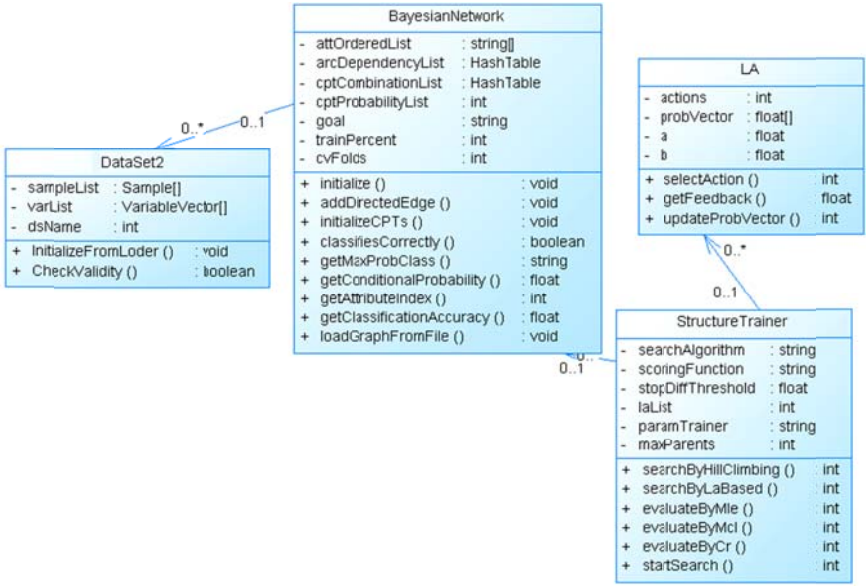

Fig7. BayesModel class diagram