Appl Intell (2016) 45:135–151
DOI 10.1007/s10489-015-0743-1

BNC-VLA: bayesian network structure learning using a team of variable-action set learning automata

S. Gheisari[1] · M. R. Meybodi[2] · M. Dehghan[2] · M. M. Ebadzadeh[2]

Published online: 1 February 2016

Abstract Bayesian Network (BN) is a probabilistic graphical model which describes the joint probability distribution over a set of random variables. One of the most important challenges in the field of BNs is to find an optimal network structure based on an available training dataset. 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 a learning automata-based algorithm has been proposed to solve the BNs structure learning problem. There is a learning automaton corresponding with each random variable and at each stage of the proposed algorithm, named BNC-VLA, a set of learning automata is randomly activated and determined the graph edges that must be appeared in that stage. Finally, the constructed network is evaluated using a scoring function. As BNC-VLA algorithm proceeds, the learning process focuses on the BN structure with higher scores. The convergence of this algorithm is theoretically proved; and also some experiments are designed to evaluate the performance of it. Experimental results show that BNC-VLA is capable of finding the optimal structure of BN in an acceptable execution time; and comparing against other search-based methods, it outperforms them.

Keywords Bayesian networks · Search and score approach · Structure training · Variable-action set learning automata

✉ S. Gheisari
so_gheisari@yahoo.com

M. R. Meybodi
mmeybodi@aut.ac.ir

M. Dehghan
dehghan@aut.ac.ir

M. M. Ebadzadeh
ebadzadeh@aut.ac.ir

[1] Department of Computer, Science and Research Branch, Islamic Azad University, Tehran, Iran

[2] Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran

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 nodes are random variables and arcs specify independent assumptions between these variables. After construction, a BN constitutes an efficient device for performing probabilistic inference [1].

One of the most important challenges in this field is training the BN that best reflects the dependence relations in a database of cases. Training an optimal structure for BNs is difficult according to the large number of possible DAG structures, given even a small number of nodes to connect. It is proven that training BN from data is an NP-Hard problem [2]. Different algorithms for training BNs from data have been developed; generally, there are two main approaches: constraint-based approach and search-and-score approach. In algorithms, following the first approach [3–7], existence of certain conditional dependencies is estimated from data. This estimation is performed using statistical or information theoretic measures. Constraints of conditional independence are propagated throughout the graph and the networks

that are inconsistent with them are eliminated from further consideration. Finally only the statistically equivalent networks consistent with conditional independence tests remain.

On the other hand, algorithms which are following the search and score approach [1], [8–15] attempt to identify the network that maximizes a score function, indicating how well the network fits the data. One such score metric is Bayesian Information Criterion, which will be explained in Section 2.1. Algorithms in this category search the space of all possible structures for the one that maximizes the score using greedy, local, or some other search algorithms.

In this paper, a new algorithm is proposed for structure training in BNs. The proposed algorithm which is named BNC-VLA, follows the search and score approach, and uses learning automata to search the optimal structure among all possible structures. Learning automata are chosen due to their learning capability and negligible computation cost. BNC-VLA is different from two prior learning automata-base algorithms, which have been proposed in [10] and [11]. Two main differences can be mentioned; firstly, in [10] and [11], learning automata are assigned to possible edges; therefore, the number of learning automata for a network with n random variables is, $\frac{1}{2}n \times (n-1)$ in [10] or $n \times (n-1)$ in [11] but in BNC-VLA learning automata are assigned to random variables, and n learning automata are used. Secondly, in [10] and [11] and also in many other algorithms, cycle removing is done in a separate phase which causes more time consuming. However, in our proposed algorithm, BNC-VLA, cycle avoidance process runs during the BN construction phase, and no more phases is needed. In order to do these the action sets of learning automata should be variable, so we have used variable-action set learning automata. At each stage, a BN is constructed based on the selected edges by learning automata; and it is evaluated using a scoring function and a training dataset. As algorithm proceeds, the learning algorithm tries to find BN structure with a higher score. Guided search by learning automata makes the search space smaller in order to find optimal BN structure in a lower computation time.

The reminder of this paper is organized as follows. Section 2 explains the BNs and structure training preliminaries. Learning automata is described briefly in Section 3. In Section 4 the proposed learning automata-based algorithm for training the BN structure is explained. Time complexity analysis and convergence results are presented in Section 5. Experimental results obtained by BNC-VLA are reported in Section 6. Finally, the paper concludes with a conclusion given in Section 7.

2 Bayesian networks and structure training

A BN describes the joint probability distribution over a set of random variables with defining series of probability independences and series of conditional independences [16]. From an informal perspective, a BN is a directed acyclic graph (DAG), where nodes are random variables, and arcs specify the independence assumptions that must be held between the random variables. Giving prior probabilities for nodes with no parent and conditional probabilities for all other nodes given all possible combinations of their parents, we can specify the probability distribution of a BN. The joint probability of any particular combination of n random variables can be written according to (1),

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(x_i | Parents\ (X_i)) \tag{1}$$

where $(X_1, \ldots, X_n)$ is the set of random variables. When the network is constructed it can be an efficient device to perform probabilistic inference. Nevertheless, the problem of constructing such a network has remained [1]. Construction of the BN is separated into two training subtasks: structure training to determine the topology of the network, and parameter training which defines numerical parameters (conditional probabilities) for a given network topology.

In this work, we focus on structure training based on search-and-score approach. Algorithms which follow this approach have two main components: a search procedure and a scoring metric. Search procedure determines an algorithm to search throughout all possible networks and then select one; and scoring metric evaluates the quality of the selected network. In the rest of this section, we briefly describe both of these components.

2.1 Scoring metric

There are varied metrics proposed for evaluating the structure of a BN such as Bayesian metric, Minimum Description Length, and Bayesian Information Criterion, to mention a few. Bayesian metric measures the quality of the BN by computing a marginal likelihood of the BN with respect to the given data and inherent uncertainties [17], [18]. Minimum Description Length 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]. And 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 metric for a constructed graph G is given by the following equation,

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

where D is the 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 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 of x_i where its parents' configuration is j, and finally N_{ijk} as the number of observation of $x_i = k$ where its parents' configuration is j, (2) can be rewritten as:

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

and in this case, BIC is converted 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 [2]. Usually, a simple greedy algorithm is used to build the network [12]. 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 [17], [19] 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 To the best of our knowledge, Genetic Algorithms [1, 8], Hill-Climbing [12], Simulated Annealing [15], A* search [13], Ant Colony optimization [14] and Learning Automata [10], [11] are used in search procedure to build the network.

3 Learning automata

A learning automaton [20], [21] is an adaptive decision-making unit that improves its performance by learning how to choose the optimal action from a finite set of allowed actions through repeated interactions with a random environment. The action is randomly chosen based on a probability distribution kept over the action-set and at each instant, the given action is served as the input to the random environment. The environment responds to the taken action in turn with a reinforcement signal. The action probability vector is updated based on the reinforcement feedback from the environment. The objective of a learning automaton is to find the optimal action from the action-set so that the average penalty received from the environment is minimized.

The environment can be described by a triple $E = \{\alpha, \beta, C\}$, where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ represents the finite set of inputs, $\beta = \{\beta_1, \beta_2, \ldots, \beta_m\}$ denotes the set of values that can be taken by the reinforcement signal, and $C = \{c_1, c_2, \ldots, c_r\}$ denotes the set of penalty probabilities, where element c_1 is associated with the given action α_1. If the penalty probabilities are constant, the random environment is said to be a stationary random environment, and if they vary with time, the environment is called a non stationary environment. The environments depending on the nature of the reinforcement signal β can be classified into P-model, Q-model and S-model. The environments in which the reinforcement signal can only take two binary values 0 and 1 are referred to as P-model environments. Another class of environments allow a finite number of values in the interval [0, 1] be taken by the reinforcement signal; such environments are referred to as Q-model environments. In S-model environments, the reinforcement signal lies in the interval [a,b].

Learning automata can be classified into two main families [20, 23] and [24]: fixed structure learning automata and variable structure learning automata. Variable structure learning automata are represented by a triple,$\langle \beta, \alpha, T \rangle$, where $\underline{\beta}$ is the set of inputs, $\underline{\alpha}$ is the set of actions, and T is learning algorithm. The learning algorithm is a recurrent relation, which is used to modify the action probability vector. Let $\alpha_i(k) \epsilon \underline{\alpha}$ and $\underline{p}(k)$ denote the action selected by a learning automaton and the probability vector defines over the action-set at instant k, respectively. At each instant k, the action probability vector $\underline{p}(k)$ is updated by the linear learning algorithm given in (4), if the selected action $\alpha_i(k)$ is rewarded by the random environment, and it is updated as given in (5) if the taken action is penalized.

$$p_j(k+1) = \begin{cases} p_j(k) + a\left[1 - p_j(k)\right] & j = 1 \\ (1-a)\, p_j(k) & \forall j \neq i \end{cases} \quad (4)$$

$$p_j(k+1) = \begin{cases} (1-b)\, p_j(k) & j = 1 \\ \left(\frac{b}{r-1}\right) + (1-b)\, p_j(k) & \forall j \neq i \end{cases} \quad (5)$$

Where a and b denote the reward and penalty parameters and determine the amount of increase and decrease of the action probabilities, respectively; and r is the number of

actions that can be taken by learning automaton. If $a = b$, the recurrent (1) and (2) are called linear reward-penalty (L_{R-P}) algorithm, if $a>b$ the given equations are called linear reward-ϵ penalty ($L_{R-\epsilon P}$), and finally if $b = 0$ they are called reward-Inaction (L_{R-I}) In the latter case, the action probability vectors remain unchanged when the taken action is penalized by the environment.

3.1 Variable action-set learning automata

A variable action-set learning automaton [20] is an automaton in which the number of actions available at each instant changes with time. Such an automaton has a finite set of n actions, $\underline{\alpha} = \{\alpha_1\alpha_2, \ldots, \alpha_r\}$. $A = \{A_1, A_2, \ldots, A_m\}$ denotes the set of action subsets and $A(k) \subseteq \alpha$ is the subset of all the actions that can be chosen by the learning automaton, at each instant k. the selection of the particular action subsets is randomly made by an external agency according to the probability distribution $\psi(k) = \{\psi_1(k), \psi_2(k), \psi_m(k)\}$ defined over the possible subsets of the actions, where $\psi_i(k) = \text{prob}[A(k) = A_i | A_i \epsilon A, 1 \leq i \leq 2^n\text{-}1]$.

$\hat{p}_i(k) = \text{prob}[\alpha(k) = \alpha_i | A(k), \alpha_i \epsilon A(k)]$ denotes the probability of choosing action α_i, conditioned on the event that the action subset $A(k)$ has already been selected and $\alpha_i \epsilon A(k)$ too. The scaled probability $\hat{p}_i(k)$ is defined as:

$$\hat{p}_i(k) = \frac{p_i(k)}{K(k)} \tag{6}$$

Where $K(k) = \sum_{\alpha_i \in A(k)} p_i(k)$ is the sum of the probabilities of the actions in subset $A(k)$, and $p_i(k) = \text{prob}[\alpha(\text{k}) = \alpha_\text{i}]$

The procedure of choosing an action and updating the action probabilities in a variable action-set learning automaton are described as follows. Let $A(k)$ be the action subset selected at instant n. Before choosing an action, the probabilities of all the actions in the selected subset are scaled as (6). Then the automaton randomly selects one of its possible actions according to the scaled action probability vector $\hat{p}_i(k)$. Depending on the received response from the environment, the learning automaton updates its scaled action probability vector. Note that only the probability of the available actions is updated. Finally, the action probability vector of the chosen subset is rescaled as $p_i(k+1) = \hat{p}_i(k+1) K(k)$, for all $\alpha_i \in A(k)$.

4 Learning automata-based BN structure training algorithms

In this section, we propose a new learning automata-based structure training algorithm for BNs, called BNC-VLA This algorithm increases the quality of constructed BNs and it decreases the execution time for constructing BNs. Overall, in comparison with other search-base algorithms like Genetic Algorithm [1, 8], Hill-Climbing [12], Simulated Annealing [15], A* search [13], Ant Colony optimization [14], and Learning Automata-based algorithms [10, 11], BNC-VLA shows superior results. In our algorithm, a team of learning automata is assigned to the random variables; they choose a network from all possible networks, and then a scoring function evaluates the chosen network. A reinforcement signal is produced based on the given score, and finally the action probability vectors of every automaton are updated according to the received signal. Pseudo code of BNC-VLA is given in Algorithm 1.

Algorithm 1 Pseudo code of BNC-VLA

1: Let n be the number of random variables, and also the number of learning automata;
2: Let k and T_k be the iteration counter and The average score of the entire constructed network until stage k, respectively. Both initially set to 0
3: Assign an automaton to each node and initially set it to the passive state.
4: **Repeat**
5: Let A denotes the set of activated automata and τ is the constructed BN which are initially null
6: **While** there are passive automata
7: Select one of the automata (passive or active with at least one unmarked action at random and call it A_i
8: **If** A_i is passive **then**
9: Insert A_i **to** A
10: **If** $|\alpha_i|$is not null **then**
11: Automaton A_i chooses one of its actions corresponding with $e_{(i,j)}$
12: Add edge $e_{(i,j)}$ to BN τ
13: chosen action is marked
14: each automaton (passive or active) prunes its action-set for cycle avoidance
15: **End If**
16: **End while**
17: Compute BIC scoring metrics of constructed BN using training dataset as BIC_k
18: **If** $BIC_k \geq T_k$**then**
19: Reward the best selected actions of activated automata
20: **Else**
21: Penalize the selected actions of activated automata
22: Update T_k using $T_k = [(\text{k-}1)T_k + BIC_k]/k$
23: Increase $k$by adding one
24: Enable all disabled actions
25: **Until** $k > Th_k$or $BIC_k \geq BIC_{opt}$

Let n denotes the number of random variables. By assigning a learning automaton to each random variable we have

a team of n learning automata $A = \{A_1, A_2, \ldots, A_n\}$ with a set of action-sets, $\underline{\alpha} = \{\underline{\alpha}_1 \underline{\alpha}_2, \ldots, \underline{\alpha}_n$ in which $\underline{\alpha}_i = \{\underline{\alpha}_{i1} \underline{\alpha}_{i2}, \ldots, \underline{\alpha}_{ij}, \ldots, \underline{\alpha}_{ir_i}$ defines the set of actions which can be taken by learning automaton A_i for each $\alpha_i \in \underline{\alpha}$ and $r_i = (n-1)$. In a network with n random variables the maximum number of directed edges from a node to other nodes is $(n-1)$; therefore, each learning automata has $(n-1)$ actions in its action set corresponding to the possible edges. Each learning automaton can be in one of two states: active and passive, all learning automata are initially set to passive state. BNC-VLA consists of a number of stages. Stage k of BNC-VLA is briefly described in the following steps:

1. BN construction

   Choose one of the passive automata at random, and mark it as active.

   Repeat the following until there is no more passive automata to be activated.

   a) The chosen automaton chooses one of its actions according to its action probability vector. Then the selected action is marked, and the automaton cannot choose it again in current stage.
   b) Each learning automaton changes its number of actions (or scales its action probability vector) by disabling the actions that may cause a cycle during the construction of the BN (the process of disabling the actions is described later).
   c) Choose one of the automata at random and if it is passive, mark it as active. Chosen automaton can be passive or active; however, active automata which have no unmarked action cannot be chosen.

2. Dynamic threshold computing

   Let us assume that BN τ_k is constructed at stage k. The average score of the entire constructed networks until stage k is computed as a dynamic threshold T_k:

$$T_k = \frac{1}{k} \sum_{i=1}^{k} BIC(\tau_i) \tag{7}$$

   where $BIC(\tau_i)$, which is defined using (3), denotes the score of the constructed network, BN τ_i in stage i.

3. Update action probability vector

   At each stage k, if the score of the constructed network, $BIC(\tau_i)$ is more than or equal to the dynamic threshold T_k, then the actions chosen by automata are rewarded and penalized otherwise. If a learning automaton has been chosen more than once, it would have more than one selected actions; however, only the best action participates in rewarding and penalizing processes. In order to find the best selected action in an automaton, for each selected edge which is associated to one action, mutual information between the nodes is computed using (8) and the training dataset.

$$I(ij) = \sum_{x_i, x_j} P\left(x_i, x_j\right) log \frac{P(x_i, x_j)}{P\left(x_i\right) P(x_j)} \tag{8}$$

   Where, $x_i x_j$ are two corresponding nodes. The best action is the action with the highest mutual information value. This approach causes to find simpler networks as well as speed up the convergence. Then, each automaton updates its action probability vector by using L_{R-I} reinforcement scheme. Disabled actions are enabled again and probability vectors are updated as described in Section 3.1 on variable action-set learning automata.

4. Termination

   Step 1, 2, and 3 are repeated until the stage number exceeds a pre-specified threshold Th_k or the score of a constructed BN becomes greater than a certain predefined score BIC_{opt}. In the latter, the network which is constructed just before the algorithm stops is the BN with the maximum expected score among all networks.

As mentioned earlier, at each stage the number of actions that can be taken by an automaton, changes (or reduces) with time in such a way that no cycle appears in the BN (see line 14 of the Algorithm). To avoid the formation of a cycle in the BN, BNC-VLA performs as follows: At each stage k each edge $e_{(ji)}$ is removed and its corresponding action is temporarily disabled in the action-set of the automaton A_j, if the edge $e_{(i,j)}$ is chosen. Then Let $\pi_{i,j}$ denotes the path connecting random variable x_i to random variable x_j, and p_j^i and q_j^i denote the choice probability of edge $e_{(i,j)}$ and path $\pi_{i,j}$, respectively. BNC-VLA also removes each edge $e_{(rs)}$ and temporarily disables its corresponding action in the action-set of the automaton A_r, if the edge $e_{(i,j)}$ is chosen and both paths $\pi_{i,r}$ and $\pi_{s,j}$ have been already constructed by the activated automata. This disabling process is repeated until the stop condition of step 1 is met. By this, the cycle freeness of the proposed BN construction algorithm is guaranteed. At each stage, the algorithm updates the action probability vectors twice; the first time is when the selected actions are rewarded or penalized, and the second time is when the disabled actions are enabled again at the end of each stage. In both cases, the action probabilities are updated as described in previous section on variable action-set learning automata.

4.1 Simple improvements

In this section, two improvements are proposed in order to speed up the convergence.

Method1: Usually, there is no prior information in learning automata, and action probability vectors are uniformly initialized. However, in the field of BNs we

can get prior information about variables' dependencies, which can be used for initializing probability vectors in order to speed up the convergence. To do this the mutual information for each possible undirected edge is computed using (8), and training dataset.

Let p_i^j be the probability of choosing action (edge) $\underline{\alpha}_{ij}$ by A_i; p_i^j can be initialized as follows.

$$p_i^j = \frac{I(i,j)}{\sum_{for\ all\ j \neq i} I(i,j)} \tag{9}$$

It is expected that the convergence speed of BNC-VLA is improved in this case.

Method2: In BNC-VLA, all learning automata use the same learning rate, which remains unchanged during the execution of the algorithm. Such a learning rate gives the equal importance to all possible edges to appear in the BN. This may prolong the convergence of the optimal solution for some learning automata and accelerate the convergence to a non-optimal solution for some others. Here, we discuss a statistical method for adjusting the learning rate to speed up the convergence rate. This method uses the Maximum Likelihood (ML) estimation in BNs for adjusting the learning rate during the algorithm. For each possible edge $e_{(i,j)}$ the probability of being observed in the BN is computed by dividing the number of samples x_j if x_i is its parent to the number of all samples. Then, the learning rate is increased for the edges which have higher ML value in comparison with others. In this case, the convergence speed of BNC-VLA may be increased.

5 Theoretical analysis

In this section, time complexity analysis and convergence results are presented.

5.1 Time complexity analysis

Lemma 1 *Let the number of iterations is iters and the number of random variables is n; the time complexity of the proposed algorithm is* $O(iter \times n^2)$.

Proof The inner loop of BNC-VLA, in Step 1, executes $n \times (n-1)$ times in the worst case and n times in the best; so the upper bound on its time complexity is $O(n^2)$ Besides, the time complexity of computing the BIC in Step 2 is $O\left(Mn^2\right)$, where M is the number of training samples; and in Step 3 the time complexity of updating the action probability vectors for n learning automata is $O(n^2)$. Finally, the outer loop is related to the number of iterations *iters*, and the time complexity of the proposed algorithm, BNC-VLA, is $O(iter \times (n^2 + Mn^2 + n^2))$; which is rewritten as: $O(iter \times n^2)$. □

5.2 Convergence results

In this section, we prove the convergence of BNC-VLA to the optimal solution, when each learning automaton updates its action-set by a linear reward-inaction reinforcement scheme. BNC-VLA is designed for stochastic environments, where the environmental parameters may vary over time; therefore, the method that is used to prove the convergence of it, partially follows the method given in [21, 23] to analyze the behavior of the learning automata operating in non-stationary environments.

Theorem 1 *Let* $q_i(k)$ *be the probability of constructing BN* τ_i*at stage k. If* $\underline{q}(k)$*is updated according to BNC-VLA, then there exists a learning rate* $a^*(\varepsilon) \in (0,1)$ *(for every* $\varepsilon > 0$*) such that for all* $a \in (0, a^*)$*, we have* $\mathbf{Prob}\left[\lim_{k\to\infty} \mathbf{q_i}(\mathbf{k}) = \mathbf{1}\right] \geq \mathbf{1} - \varepsilon$

Proof The steps of convergence proof are briefly outlined as follows. At first, it is proved that the penalty probability of each BN converges to the constant value of the final penalty probability, if k is selected large enough. This property is shown in Lemma 2. Then, it is shown that the probability of choosing a BN with the maximum score is a sub-martingale process for large values of k, and so the changes in the probability of constructing the BN are always nonnegative. Lemma 3 and 4 show this result. Finally, the convergence of BNC-VLA to a BN with the maximum expected score is proved by using martingale convergence theorems. Therefore, the following lemmas need to be proved before starting the proof of Theorem 1. □

Lemma 2 *If BN* τ_i *penalized with probability* $c_i(k)$ *at stage* k *(i.e.* $c_i(k) = prob\left[\overline{BIC}\tau_i < T_k\right]$*) and* $\lim_{k\to\infty} c_i(k) = c_i^*$*. Then, for every* $\varepsilon \in (0,1)$ *and* $k > K(\varepsilon)$ *we have,* $prob\left[\left|c_i^* - c_i(k)\right| > 0\right] < \varepsilon$

Proof Let c_i denotes the final value of probability $c_i(k)$ when k is large enough. Using weak law of large numbers, we have concluded that $prob\left[\left|c_i^* - c_i(k)\right| > \varepsilon\right] \to 0$

Hence, for every $\varepsilon \in (0,1)$, there exists $a^*(\varepsilon) \in (0,1)$ and $K(\varepsilon) < \infty$ such that for all $a < a^*$ and $k > K(\varepsilon)$ we have $prob\left[\left|c_i^* - c_i(k)\right| > 0\right] < \varepsilon$, and the proof of Lemma 2 is completed. □

Lemma 3 *Let* $c_j(k) = prob[\overline{BIC}\tau_j(k+1) < T_k$ *and* $d_j(k) = 1 - c_j(k)$ *be the probability of penalizing and*

rewarding BN τ_j at stage k, respectively. If $\underline{q}(k)$ evolves according to BNC-VLA, then the conditional expectation of $q_i(k)$ is defined as

$$E[q_i(k+1)|q(k)] = \sum_{j=1}^{r} q_j(k)\left[c_j(k)\, q_i(k) + d_j \prod_{e_{(m,n)}\in\tau_i} \delta_n^m(k)\right]$$

Where

$$\delta_n^m(k) = \begin{cases} p_n^m(k+1) = p_n^m(k) + a\left(1 - p_n^m(k)\right); & e_{(m,n)} \in \tau_j \\ p_n^m(k+1) = p_n^m(k)\,(1-a); & e_{(m,n)} \notin \tau_j \end{cases}$$

Where r denotes all constructed BNs.

Proof Since the reinforcement scheme that is used to update the probability vectors in BNC-VLA is L_{R-I}, at each stage k the probability of choosing the BN τ_i (i.e., $q_i(k)$) remains unchanged with probability $c_j(k)$ (for all $j = 1, 2, \ldots, r$) when the selected BN τ_j is penalized by the random environment. On the other hand, when the selected BN τ_j is rewarded, the probability of choosing edges of BN τ_i increases by a given learning rate as that of the other edges decreases Hence the lemma is proven. □

Lemma 4 *The increment in the conditional expectation of $q_i(k)$ is always non-negative subject to $\underline{q}(k)$ is updated according to BNC-VLA. That is, $\Delta q_i(k) > 0$.*

Proof Define $\Delta q_i(k) = E[q_i(k+1)|q(k)] - q_i(k)$

From Lemma 3, we have

$$\begin{aligned} \Delta q_i(k) &= E[q_i(k+1)|q(k)] - q_i(k) \\ &= \sum_{j=1}^{r} q_j(k)\,[\, c_j(k)\, q_i(k) \\ &\quad + d_j \prod_{e_{(m,n)}\in\tau_i} \delta_n^m(k)\,] - q_i(k) \end{aligned} \tag{10}$$

where

$$\delta_n^m(k) = \begin{cases} p_n^m(k+1) = p_n^m(k) + a(1 - p_n^m(k)); & e_{(m,n)} \in \tau_j \\ p_n^m(k+1) = p_n^m(k)\,.\,(1-a); & e_{(m,n)} \notin \tau_j \end{cases} \tag{11}$$

$p_n^m(k)$ is the probability of choosing edge $e_{(n,m)}$ at stage k. The probability with which the BNs are constructed, rewarded or penalized is defined as the result of the probability of choosing the edges along the BNs, so we have

$$\Delta q_i(k) = \sum_{j=1}^{r} \prod_{e_{(m,n)}\in\tau_j} p_n^m(k) \left[\prod_{e_{(m,n)}\in\tau_j} c_n^m(k) \prod_{e_{(m,n)}\in\tau_i} p_n^m(k) + \prod_{e_{(m,n)}\in\tau_j} d_n^m(k) \prod_{e_{(m,n)}\in\tau_i} \delta_n^m(k) \right] - \prod_{e_{(m,n)}\in\tau_i} p_n^m(k)$$

Where $\delta_n^m(k)$ is defined as given in (10) $c_n^m(k)$ is the probability of penalizing edge $e_{(m,n)}$ at stage k, and $d_n^m(k) = 1 - c_n^m(k)$. At each stage, BNC-VLA chooses edges of DAG constructing one of r possible BNs.

$$\Delta q_i(k) = \prod_{e_{(m,n)}\in\tau_i} E\left[p_n^m(k+1)\,|p^m(k)\right] - \prod_{e_{(m,n)}\in\tau_i} p_n^m(k)$$

The above mentioned equality can be rewritten as:

$$\begin{aligned} \Delta q_i(k) &\geq \prod_{e_{(m,n)}\in\tau_i} E\left[p_n^m(k+1)\,|p^m(k)\right] - p_n^m(k) \\ &= \prod_{e_{(m,n)}\in\tau_i} \Delta p_n^m(k) \end{aligned} \tag{12}$$

And $\Delta p_n^m(k) =$a. $p_n^m(k) \sum_{s\neq n}^{r_m} p_s^m(k)\,.(c_s^m(k) - c_n^m(k))$

$q_i(k) \in (0,1)$ for all $\underline{q} \in S_r^0$, where $S_r = \{\underline{q}(k) : 0 \leq q_i(k) \leq 1;\ \sum_{i=1}^{r} q_i(k) = 1\}$ and S_r^0 denotes the interior of S_r. Hence, $p_n^m(k) \in (0,1)$ for all m, n. Since edge $e_{(m,n)} \in \tau_i$ is the edge with high probability which can be selected by automaton A_m, it is shown that $c_s^{m^*}(k) - c_n^{m^*}(k) > 0$ for all $s \neq n$. It follows from Lemma 2 that for large values of k, $c_s^m(k) - c_n^m(k) > 0$. Therefore, we conclude that for large values of k,the right hand side of the above equation consists of nonnegative quantities and so we have $\prod_{e_{(m,n)}\in\tau_i} a\, p_n^m(k) \sum_{s\neq n}^{r_m} p_s^m(k)\,.(c_s^m(k) - c_n^m(k)) \geq 0$ and from (10), we have

$$\Delta q_i(k) \geq \prod_{e_{(m,n)}\in\tau_i} a\, p_n^m(k) \sum_{s\neq n}^{r_m} p_s^m(k)\,. \\ (c_s^m(k) - c_n^m(k))$$

This completes the proof of this lemma. □

Corollary 1 *The set of unit vectors in $S_r - S_r^0$ forms the set of all absorbing barriers of the Markov process $\{\underline{q}(k)\}_{k\geq 1}$, where $S_r^0 = \left\{\underline{q}(k) : q_i(k) \in (0,1);\ \sum_{i=1}^{r} q_i(k) = 1\right\}$.*

Proof Lemma 4 implicitly proves that $\underline{q}(k)$ is a submartingale. Using martingale theorems and the fact that $\underline{q}(k)$ is a non-negative and uniformly bounded function, it is concluded that $q_i(k)$ converges to $q*$ with probability one. Hence, from (10), it can be seen that $q_i(k+1) \neq q_i(k)$ with a nonzero probability if and only if $q_i(k) \notin 0, 1\}$, and $\underline{q}(k+1) = \underline{q}(k)$ with probability one if and only if $q^* \in \{0, 1\}$ where $\lim_{k\to\infty} q_i(k) = q^*$, and hence the proof is completed. □

Let $\Gamma_i(q)$ be the probability of convergence of BNC-VLA to unit vector e_i with initial probability vector; $\underline{q}$. $\Gamma_i(q)$ is defined as follows:

$$\begin{aligned}\Gamma_i(q) &= prob[q_i(\infty)=1|\underline{q}(0)=\underline{q}]\\ &= prob[q^*=e_i|\underline{q}(0)=\underline{q}].\end{aligned}$$

Let $C(S_r): S_r \to \Re$ be the state space of all real-valued continuously differentiable functions with bounded derivative defined on S_r, where $\Re$ is the real line. If $\psi(.) \in C(S_r)$, the operator U is defined as:

$$U\psi(q) = E[\psi(q(k+1))|q(k)=q] \tag{13}$$

where $E[.]$ represents the mathematical expectation.

It has shown in [23] that operator U is linear, and it preserves the non-negative functions as the expectation of a non-negative function remains nonnegative. In other words, $U\psi(q) \geq 0$ for all $q \in S_r$, if $\psi(q) \geq 0$. If the operator U is applied n (for all $n>1$) times repeatedly, we have

$$U^{n-1}\psi(q) = E[\psi(q(k+1))|q(1)=q].$$

Function $\psi(q)$ is called super-regular (sub-regular) if and only if $\psi(q) \geq U\psi(q)$ $(\psi(q) \leq U\psi(q))$, for all $q \in S_r$. It has been shown in [23] that $\Gamma_i(q)$ is the only continuous solution of $U\Gamma_i(q) = \Gamma_i(q)$ subject to the following boundary conditions.

$$\Gamma_i(e_i) = 1$$

$$\Gamma_i(e_j) = 0;\ \ j \neq i \tag{14}$$

Define $\phi_i[x,q] = \frac{e^{-xq_i/a}-1}{e^{-x/a}-1}$ where $x>0$. $\phi_i[x,q] \in C(S_r)$ satisfies the boundary condition above.

Theorem 2 *Let $\psi(.) \in C(S_r)$ be super-regular with $\psi_i(e_i)=1$ and $\psi_i(e_j)=0$ for $j \neq i$ then*

$$\psi_i(q) \geq \Gamma_i(q)$$

for all $q \in S_r$. If $\psi(.) \in C(S_r)$ is sub-regular with the same boundary conditions, then

$$\psi_i(q) \leq \Gamma_i(q) \tag{15}$$

for all $q \in S_r$.

Proof Theorem 2 has been proved in [21].

In what follows, we show that $\phi_i[x,q]$ is sub-regular function, and $\phi_i[x,q]$ qualifies as a lower bound on $\Gamma_i(q)$. Super and sub-regular functions are closed under addition and multiplication by a positive constant, if and only if $\phi(.)$ is super regular then $-\phi(.)$ is sub-regular. Therefore, it follows that $\phi_i[x,q]$ is sub-regular if and only if $\theta_i[x,q] = e^{-xq_i/a}$ is super-regular.

We now determine the conditions under which $\theta_i[x,q]$ is super-regular. From the definition of operator U given in (13), we have:

$$\begin{aligned}&E\left[e^{\frac{xq_i(k+1)}{a}}\middle|q(k)=q\right]\\ &=\left[\sum_{j=1}^{r} q_j d_j^* e^{-\frac{x}{a}\left[\prod_{\substack{e_{(m,n)}\in\tau_i,\\ e_{(m,n)}\in\tau_j}}(p_n^m+a(1-p_n^m))\right]}\right.\\ &\quad\left.+\sum_{j=1}^{r} q_j d_j^* e^{-\frac{x}{a}\left[\prod_{\substack{e_{(m,n)}\in\tau_i,\\ e_{(m,n)}\notin\tau_j}}(p_n^m(1-a))\right]}\right]\\ &U\theta_i(x,q)\\ &=\left[q_j d_j^* e^{-\frac{x}{a}(q_i+a(1-q_i))}\right.\\ &\quad+\sum_{j=1}^{r} d_j^* e^{-\frac{x}{a}\left[\prod_{\substack{e_{(m,n)}\in\tau_i,\\ e_{(m,n)}\in\tau_j}}(p_n^m+a(1-p_n^m))\right]}\\ &\quad\left.+\sum_{j\neq 1}^{r} q_j d_j^* e^{-\frac{x}{a}\left[\prod_{\substack{e_{(m,n)}\in\tau_i,\\ e_{(m,n)}\notin\tau_j}}(p_n^m(1-a))\right]}\right]\end{aligned}$$

Where d_j^* denotes the final value to which the reward probability $d_j(k)$ is converged (for large value of k), and $e^{-\frac{x}{a}(q_i+a(1-q_i))}$ is the expectation of $\theta_i(x,q)$ when the BN τ_i is rewarded by the environment.

$$\begin{aligned}U\theta_i(x,q) &= \left[q_j d_j^* e^{-\frac{x}{a}(q_i+a(1-q_i))}\right.\\ &\quad\left.+\sum_{j\neq i} q_j d_j^* e^{-\frac{x}{a}\left(q_i(1-a)\prod_{\substack{e_{(m,n)}\in\tau_i,\\ e_{(m,n)}\in\tau_j}}\frac{(p_n^m+a(1-p_n^m))}{(p_n^m(1-a))}\right)}\right]\\ &= \left[\sum_{j=i} q_j d_j^* e^{-\frac{x}{a}\rho_j^i(q_i+a(1-q_i))}\right.\\ &\quad\left.+\sum_{j\neq i} q_j d_j^* e^{-\frac{x}{a}\rho_j^i(q_i(1-a))}\right]\end{aligned}$$

Where $\rho_j^i > 0$ is defined as

$$\rho_j^i = \begin{cases}\prod_{\substack{e_{(m,n)}\in\tau_i,\\ e_{(m,n)}\in\tau_j}} \frac{(p_n^m+a(1-p_n^m))}{(p_n^m(1-a))}; & i \neq j\\ 1; & i=j \text{ or } (\tau_i \cap \tau_j)=\emptyset\end{cases}$$

$$
\begin{aligned}
&U\theta_i(x,q)-\theta_i(x,q)\\
&\quad=\Big[e^{-xq_i\rho_j^i/a}\sum\nolimits_{j=i}q_i d_j^* e^{-x(1-q_i)\rho_j^i}\\
&\qquad+e^{-xq_i\rho_j^i/a}\sum\nolimits_{j\neq i}q_i d_j^* e^{xq_i\rho_i^j}\Big]-e^{-xq_i/a}
\end{aligned}
$$

$\theta_i(x,q)$ is super-regular if

$$
\begin{aligned}
&e^{-xq_i\rho_j^i/a}\sum\nolimits_{j=i}q_i d_j^* e^{-x(1-q_i)\rho_j^i}\\
&\quad+e^{-xq_i\rho_j^i/a}\sum\nolimits_{j\neq i}q_i d_j^* e^{xq_i\rho_i^j}\le e^{xq_i/a}
\end{aligned}
$$

and

$$
U\theta_i(x,q)\le e^{xq_i/a}q_i d_j^* e^{-x(1-q_i)}+e^{xq_i/a}\sum_{j\neq i}q_i d_j^* e^{xq_i},
$$

if $\theta_i(x,q)$ is super-regular. Therefore, we have

$$
\begin{aligned}
&U\theta_i(x,q)-\theta_i(x,q)\\
&\le\Big[e^{-xq_i/a}q_i d_j^* e^{-x(1-q_i)}+e^{-xq_i/a}\sum\nolimits_{j\neq i}q_i d_j^* e^{xq_i}\Big]-e^{-xq_i/a}.
\end{aligned}
$$

After multiplying and dividing the right hand side of the inequality above by $-xq_i$ and some algebraic simplifications, we have

$$
\begin{aligned}
&U\theta_i(x,q)-\theta_i(x,q)\\
&\le -xq_i e^{-xq_i/a}\left[q_i d_j^*\frac{e^{-x(1-q_i)}-1}{-xq_i}-\sum\nolimits_{j\neq i}q_i d_j^*\frac{e^{xq_i}-1}{xq_i}\right]\\
&=-xq_i e^{-\frac{xq_i}{a}}\left[d_j^*\frac{e^{-x(1-q_i)}-1}{-x}-\sum\nolimits_{j\neq i}q_i d_j^*\frac{e^{xq_i}-1}{xq_i}\right]\\
&=-xq_i e^{-xq_i/a}\left[(1-q_i)d_j^*\frac{e^{-x(1-q_i)}-1}{-x(1-q_i)}-\sum\nolimits_{j\neq i}q_i d_j^*\frac{e^{xq_i}-1}{xq_i}\right]
\end{aligned}
$$

and

$$
V[u]=\begin{cases}\frac{e^u-1}{u};& u\neq 0\\ 1; & u=0\end{cases}
$$

$$
\begin{aligned}
&U\theta_i(x,q)-\theta_i(x,q)\\
&\quad\le -xq_i e^{-\frac{xq_i}{a}}\Big[(1-q_i)\,d_i^* V[-x(1-q_i)]\\
&\qquad-\Big(\sum\nolimits_{j\neq i}q_j d_j^*\Big)V[xq_i]\Big]=xq_i\theta_i(x,q)\,G_i(xq)
\end{aligned}
$$

where

$$
\begin{aligned}
G_i(x,q)\;=\;&(1-q_i)\,d_i^* V[-x(1-q_i)]\\
&-\Big(\sum\nolimits_{j\neq i}q_j d_j^*\Big)V[xq_i]
\end{aligned}
\tag{16}
$$

Therefore, $\theta_i(x,q)$ is super-regular if

$$
G_i(x,q)\ge 0. \tag{17}
$$

From (16), it follows that $\theta_i(x,q)$ is super-regular if we have

$$
f_i(x,q)=\frac{V[-x(1-q_i)]}{V[xq_i]}\le\frac{\sum_{j\neq i}q_i d_i^*}{(1-q_i)d_i^*}. \tag{18}
$$

The right hand side of the inequality (18) consists of the non-negative terms, so we have

$$
\begin{aligned}
\Big(\sum\nolimits_{j\neq i}q_i\Big)\min\nolimits_{j\neq i}\left(\frac{d_j^*}{d_i^*}\right)&\le\frac{1}{(1-q_i)}\sum\nolimits_{j\neq i}q_j\frac{d_j^*}{d_i^*}\\
&\le\Big(\sum\nolimits_{j\neq i}q_i\Big)\max\nolimits_{j\neq i}\left(\frac{d_j^*}{d_i^*}\right).
\end{aligned}
$$

Substituting $\sum_{j\neq i}q_i$ by $(1-q_i)$ in the above inequality, we can rewrite it as:

$$
\min\nolimits_{j\neq i}\left(\frac{d_j^*}{d_i^*}\right)\le\frac{\sum_{j\neq i}q_j\frac{d_j^*}{d_i^*}}{\sum_{j\neq i}q_j}\le\max\nolimits_{j\neq i}\left(\frac{d_j^*}{d_i^*}\right).
$$

From (18), it follows that $\theta_i(x,q)$ is super-regular if we have

$$
f_i(x,q)\ge\max\nolimits_{j\neq i}\left(\frac{d_j^*}{d_i^*}\right)
$$

For further simplification, let us employ logarithms. If $\Delta(q,x)=lnf_i(x,q)$, it has been shown in [23] that

$$
-\int_0^x H'(u)\,du\le\Delta(q,x)\le-\int_{-x}^0 H'(u)\,du
$$

$$
H(u)=\frac{dH(u)}{du},\;H(u)=lnV(u).
$$

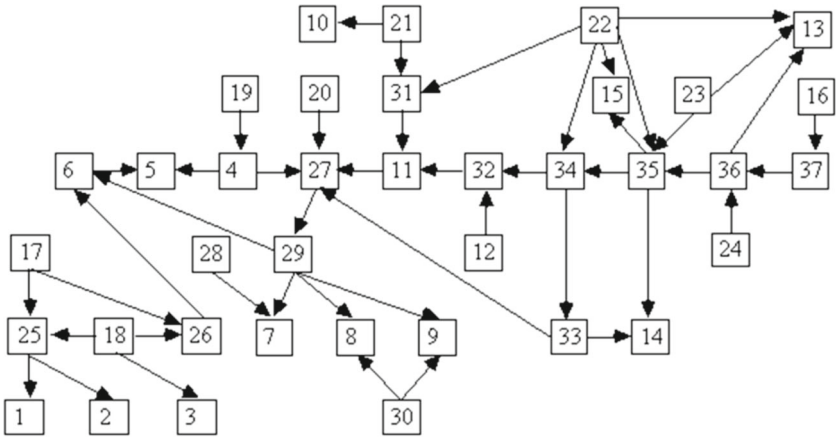
Therefore, we have

$$
\frac{1}{V[x]}\le\frac{V[-x(1-q_i)}{V[xq_i]}\le V[-x]
$$

and

$$
\frac{1}{V[x]}=\max\nolimits_{j\neq i}\left(\frac{d_j^*}{d_i^*}\right). \tag{19}
$$

Fig. 1 The ALARM network



Let $x*$ be the value of x for which (19) is true. It is shown that there exists a value of $x>0$ under which (19) is satisfied, if (d_j/d_i) is smaller than 1 for all $j \neq i$. By choosing $x = x*$, (19) holds true. Consequently, (15) is true and $\theta_i(x, q)$ is a super-regular function. Therefore, $\phi_i[x, q] = \frac{e^{-xq_i/a}-1}{e^{-x/a}-1}$ is a sub-regular function satisfying the boundary conditions given in (14). From Theorem 2 and inequality (15), we conclude that

$$\phi_i[x, q] \leq \Gamma_i(q) \leq 1.$$

From definition of $\phi_i[x, q]$, we see that given any $\varepsilon>0$ there exists a positive constant $a*<1$ such that $1 - \varepsilon \leq \phi_i[x, q] \leq \Gamma_i(q) \leq 1$ for all $0 < a \leq a^*$.

Thus we conclude that the probability with which BNC-VLA constructs the BN with the maximum BIC is equal to 1 as $k \to \infty$, and so Theorem 1 is proved. □

Theorem 3 *Let $q_i(k)$ be the probability of constructing BN τ_i at stage k, and (1-ε) be the probability with which Algorithm 1 converges to BN τ_i. If $q(k)$ is updated by Algorithm 1, then for every error parameter $\varepsilon \in (0, 1)$ there exists a learning rate $a \in (\varepsilon q)$ so that $\frac{xa}{e^{xa}-1} = \left(\frac{d_j}{d_i}\right)$, where $1 - e^{-xq_i} = \left(1 - e^{-x}\right)(1 - \varepsilon)$ and $q_i = [q_i(k)|k = 0]$.*

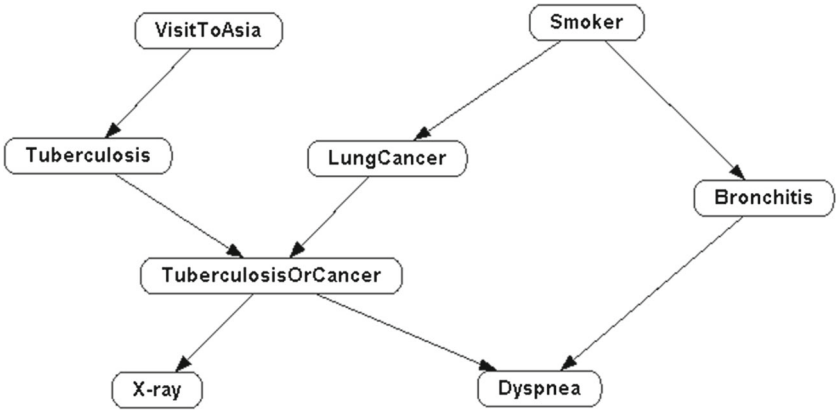
Proof It has been proved in [23] that there always exists a $x>0$ under which (19) is satisfied, if $\frac{d_j}{d_i} < 1$ for all $j \neq i$. Hence, it is concluded that $\phi_i[x, q] \leq \Gamma_i(q) \leq \frac{1-e^{-xq_i}}{1-e^{-x}}$ where q_i is the initial choice probability of the optimal BN τ_i. From Theorem 1, for each $0<a<a*$ the probability of converging Algorithm 1 to the BN with the maximum expected BIC is (1- ε) where $a^* \in (0, 1)$. Therefore, it is concluded that,

$$\frac{1 - e^{-xq_i}}{1 - e^{-x}} = 1 - \varepsilon. \tag{20}$$

It is shown that for every error parameter $\varepsilon \in (0, 1)$there exists a value of x under which (19) is satisfied, and so we have $\frac{x^*a}{e^{x^*a}-1} = \max_{j\neq i}\left(\frac{d_j^*}{d_i^*}\right)$.

It is concluded that for every error parameter $\varepsilon \in (0, 1)$ there exists a learning rate $a \in (\varepsilon q)$ under which the probability of converging Algorithm 1 to the BN with the

Fig. 2 The ASIA network



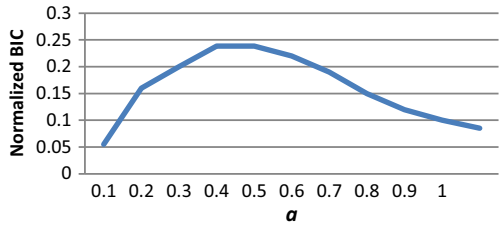


Fig. 3 Analysis the effect of parameter a for BNC-VLA

maximum expected BIC is greater that (1-ε) and hence the theorem is proved. □

6 Numerical results

In order to evaluate the performance of BNC-VLA, two classes of experiments have been developed. First we have determined two well-known BNs (structures + conditional probabilities). We have simulated them, using BNC-VLA and databases of cases, which reflect the conditional independence relations between the variables. We have evaluated the constructed BNs; and we have also compared the proposed algorithm, BNC-VLA with other algorithms in this case. Second, since classification is one of the important applications of BNs, for 25 chosen datasets the classification accuracy of the proposed algorithm has been compared against other algorithms.

6.1 Evaluation results on well-known networks

The BNs used in these experiments are the ALARM and the ASIA networks. ALARM network [25] is a medical diagnostic alarm message system for patient monitoring; it contains 37 nodes and 46 arcs (see Fig. 1). Researchers in this field use datasets which are generated from three versions of ALARM network with the same structure but different probability distributions. We use the 5000 first cases from the database that was generated by Edward Herskovits [26]. ASIA network [27] is a simple network with eight binary nodes and eight arcs (Fig. 2). It was introduced by Lauritzen and Spiegelhalter to illustrate their method of propagation of evidence, considers a small piece of factious qualitative medical knowledge. A dataset with 5000 cases is sampled using the Netica tool [28] for constructing its network.

6.1.1 Sensitivity analysis

In this section, we study the effect of learning parameters, a in (4) and $b$in (5), on the performance of the BNC-VLA Since the reward-Inaction (L_{R-I}) algorithm has been used, $b = 0$, and we only peruse the effect of parameter a in order to find the best value of it in our implementation. To

Table 1 Average results of BNC-VLA after 10 runs

Number of cases	500		1500		3000		5000	
Parameters	ALARM	ASIA	ALARM	ASIA	ALARM	ASIA	ALARM	ASIA
BIC	398228±12.61	9357.22±7.12	15461.20±6.64	10025.62±4.54	23800.06±1.45	11328.17±0.21	23853.83±0.72	12145.87±0.2
	(3998.78)	(9361.78)	(15466.73)	(10023.01)	(23801.04)	(11328.26)	(23854.23)	(12146.00)
Hamming Distance	17.7±4.75	3.8±2.97	4.1±2.0	1.3±0.7	2.0±0.0	0.0±0.0	2.0±0.0	0.0±0.0
	(13)	(0)	(2)	(0)	(2)	(0)	(2)	(0)
Computation time	71.30±22.63	24.1±12.05	81.19±28.12	27.6±12.54	96.52±18.68	30.22±18.33	116.14±16.35	35.5±20.06

Table 2 Comparative experiments results after 10 runs

Algorithms	ALARM BIC	Normalized[1] Hamming Distance	Computation time	ASIA BIC	Normalized Hamming Distance	Computation time
BNC-VLA	23853.83± 0.72 (23854.23)	1.0	116.14	12145.87± 0.2 (12146.00)	1.0	35.5
FALA-based	15472.14± 27.10 (15501.12)	2.05	195.14	10494.75± 17.02 (10509.99)	1.0	71.12
Hill climbing-based	8781.2± 35.44 (8802.32)	7.21	308.0	8065.42± 16.13 (8075.03)	2.75	100.54
Genetic algorithm-based	11461.15±16.21 (11475.57)	2.82	284.15	9839.18± 8.70 (9845.22)	1.25	90.3
ACO	11822.67± 8.98 (11827.98)	2.93	279.12	9752.55± 4.06 (9755.55)	1.30	89.07

do this the database with 5000 cases is considered to construct the ALARM network. Figure 3 represents the results. It indicates that the best value of parameter a is between 0.4 and 0.5, therefore in following experiments the value of a is 0.4.

6.1.2 Performance evaluation of BNC-VLA

In these experiments, BNC-VLA is employed to construct well-known BNs, the ALARM and the ASIA, using their training samples. The constructed network for ALARM is identical to its origin; 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. On the other hand, the constructed BN for ASIA is completely the same with its original network.

Then, we consider different subsets of datasets for both ALARM and ASIA networks, which consist of the first 500, 1500, 3000, and 5000 cases. In order to evaluate the behavior of the algorithm, for each dataset of cases, following parameters are considered:

I. BIC as scoring metric
II. Hamming distance
III. Computation time

BIC is measured by (3). The interpretation of BIC is: the higher this parameter, the better the network. Hamming distance directly compares the structure of the learned and

Table 3 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	sat image	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				

Table 4 Average error rate for different datasets

	BNC-VLA	FALA	ACO	Genetic	Hill climbing	NB	TAN
Australian	0.1221	0.1236	0.1488	0.1444	0.1391	0.1489	0.1751
Breast	0.0201	0.0425	0.0339	0.0351	0.0668	0.0245	0.0351
Chess	0.0312	0.0422	0.06	0.062	0.0966	0.1266	0.076
Cleve	0.1664	0.1997	0.166	0.1704	0.182	0.1791	0.2164
Corral	0.0041	0.0119	0.1273	0.1271	0.0114	0.1277	0.0143
Crx	0.1392	0.158	0.1505	0.1503	0.137	0.1505	0.1631
Diabetes	0.2168	0.2666	0.2419	0.2384	0.255	0.2571	0.2384
Flare	0.1803	0.1814	0.1825	0.178	0.181	0.2024	0.1789
German	0.2412	0.2643	0.2456	0.2609	0.3085	0.2458	0.2609
Glass	0.3012	0.4173	0.422	0.4818	0.4412	0.4412	0.4578
Glass2	0.1634	0.2691	0.1938	0.1949	0.2329	0.2236	0.2249
Heart	0.1526	0.1874	0.155	0.1547	0.2221	0.155	0.1847
Hepatitis	0.0938	0.1618	0.1294	0.13	0.1967	0.193	0.1302
Iris 0.0401	0.042	0.0485	0.0463	0.0414	0.0699	0.0763	
Letter	0.023	0.183	0.3068	0.3752	0.1896	0.3068	0.1752
Lymphography	0.1396	0.1635	0.147	0.1484	0.2247	0.1662	0.1784
Mofn-3-7-10	0.08	0.0859	0.1367	0.1932	0.0859	0.1328	0.085
Pima	0.1375	0.2666	0.2505	0.2484	0.255	0.2571	0.2384
Sat image	0.1481	0.1745	0.173	0.162	0.184	0.1915	0.1395
Segment	0.0544	0.0571	0.0701	0.0671	0.0831	0.1221	0.0675
Shuttle-small	0.004	0.0047	0.0083	0.0082	0.0145	0.014	0.0093
Soybean-large	0.0629	0.0754	0.092	0.0963	0.0922	0.0852	0.0644
Vehicle	0.2736	0.2922	0.3453	0.4327	0.3451	0.3892	0.2718
Vote	0.0319	0.0417	0.037	0.0387	0.0467	0.0991	0.0509
Waveform-21	0.1770	0.267	0.1772	0.1734	0.2543	0.2142	0.2534
average	0.120174	0.159136	0.161916	0.172716	0.171472	0.18094	0.1586

the original networks. We define the Hamming distance between two DAGs as the number of following operators required to make the DAGs match: add, remove, or reverse a directed edge. The lower Hamming distance indicates the more similarity between the constructed network and the original one. BNC-VLA is also evaluated based on the computation time; to do this it is implemented 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 computation time, BNC-VLA is run with no prior limitation in the number of iterations until more repetition does not increase the score.

Table 1 represents the average results found after 10 independent runs with the databases of 500, 1500, 3000, and 5000 cases. In this table $\mu + \sigma$ indicates the mean, and the standard deviation over the executions carried out. The value inside (.) is the best resul found along the experimentation. As the results indicate, BNC-VLA constructs satisfactory networks (networks with low Hamming distance and high BIC score) especially with databases of 3000 and 5000 cases, in acceptable time consuming. Better results for ASIA were predictable, because the ASIA is simpler than the ALARM.

6.1.3 Comparison the BNC-VLA with other algorithms

In the following experiments, BNC-VLA is compared against other algorithms. All implemented algorithms are described below:

- Other learning automata-based method, which uses a team of FALA [10];
- Hill climbing-based algorithm [12];
- Genetic algorithm which, firstly, findsthe best ordering of variables and then starts the search process [3];
- And finally Ant colony optimization [14] called ACO.

Many other BN structure learning algorithms exist; A* search-based algorithm with a shortest path perspective [13] is an example. But, it is not possible and necessary to compare our proposed algorithms with all of them; so we have selected more comprehensive algorithms, which also stand in the same class with our algorithm.

Datasets with 5000 cases are considered for both the ALARM and the ASIA networks; considered metrics are the same as the metrics in previous experiments. Algorithms have run with no prior limitation in time until no improvement in the score is observed. Table 2 shows the results after

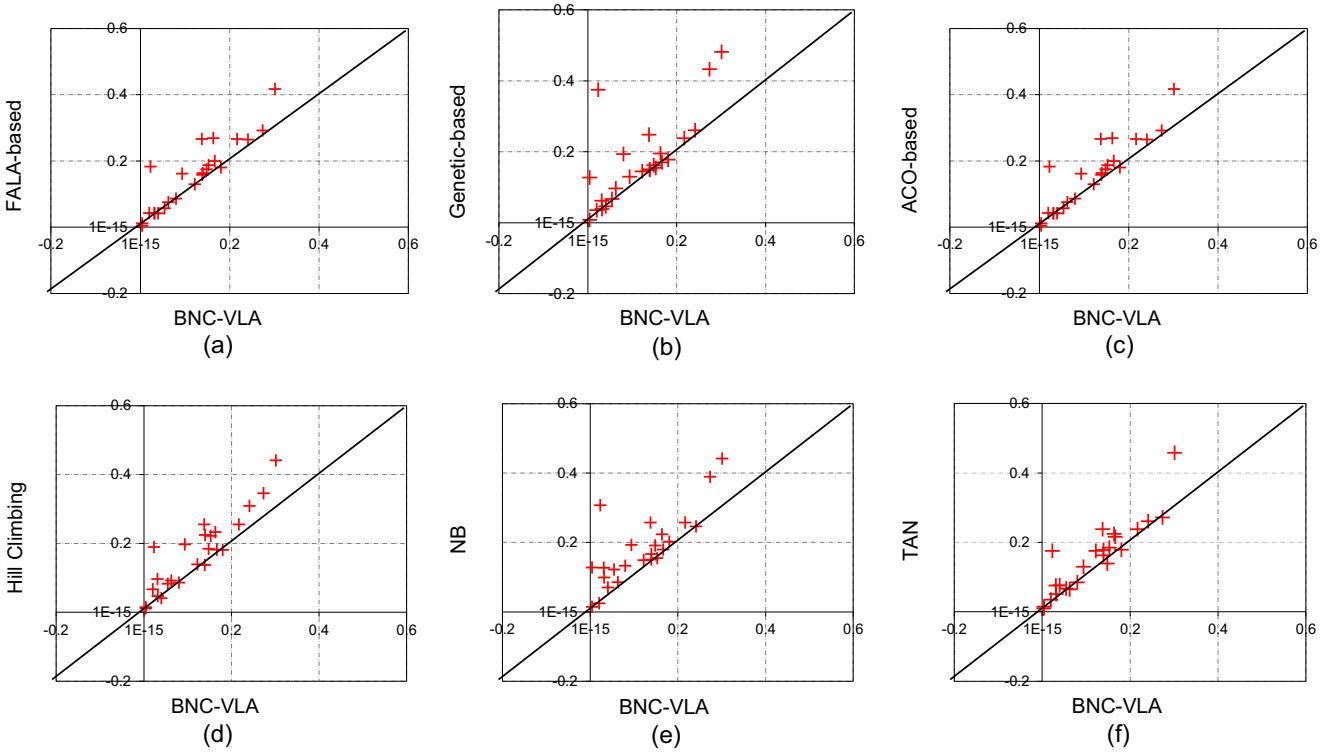


Fig. 4 Scatter chart, compare BNC-VLA classifier with (**a**) FALA-based, (**b**) Genetic-based, (**c**) ACO-based, (**d**) Hill Climbing-based (**e**) Na¿ve Bayes, and (**f**) TAN, classifiers; points above y = x show better performance of proposed algorithm

10 independent runs. In this table $\mu + \sigma$ indicates the mean, and the standard deviation over the executions carried out. The value inside (.) is the best result found along the experimentation. From the results, we observe that the BNC-VLA is superior to other algorithms in both the ALARM and the ASIA networks.

6.2 Evaluating the predictive ability of proposed algorithm in classification

A constructed BN is an efficient device to perform probabilistic inference whereupon, predictive ability in different applications is one of the significant 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, and disease diagnosis for assessing patients' breast cancer risk [29]. In this section, at first we briefly explain about BNs classifiers in Section 6.2.1, and then by choosing datasets of different applications, we evaluate the classification accuracy and classification time of different algorithms in Sections 6.2.2 and 6.2.3, respectively.

6.2.1 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 test samples (classification accuracy), the correct prediction for $(x_1, x_2 \ldots x_{v-1})$ is a class that maximizes $P(c|x_1, x_2 \ldots x_{v-1})$. If there is a BN

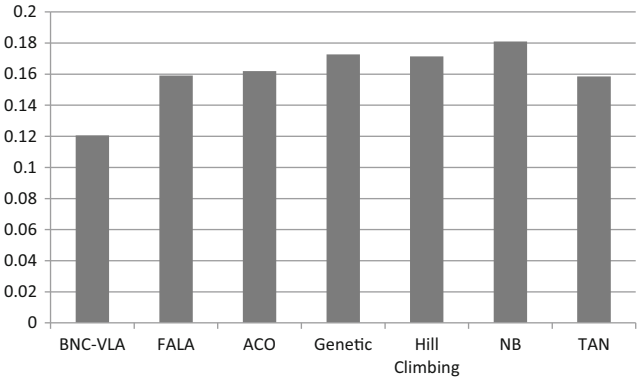


Fig. 5 Average error rate for classification

Table 5 Classification execution time of different algorithms on different datasets

	BNC-VLA	FALA	ACO	Genetic	Hill climbing	NB	TAN
Letter	136.09	326.49	628.64	652.13	898.81	676.42	458.59
Chess	71.93	165.93	349.88	350.03	570.22	357.36	288.45
German	33.12	58.12	136.32	132.46	164.80	148.42	95.45
Hepatitis	21.99	44.99	59.34	52.85	93.01	66.54	88.43
Breast	13.42	21.42	55.13	50.63	241.09	82.73	28.43
Glass2	7.14	14.14	29.14	29.75	49.41	28.45	24.39
Iris	4.32	6.42	17.12	17.83	29.29	19.90	14.54

over $(x_1, x_2 \ldots x_{v-1}, C)$, we could 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.

6.2.2 Comparison the classification accuracy of BNC-VLA against other classifiers

In order to evaluate and compare the classification accuracy of proposed algorithm, BNC-VLA, 25 datasets are used, which consist of 21 datasets from UCI [30] and others from [31]. Table 3 shows a brief description of these datasets. All other implemented classifiers are described as follows:

- Naïve Bayes classifier;
- TAN-based classifier;
- FALA-based classifier[10];
- Hill climbing-based classifier [12];
- Genetic algorithm-based classifier [3];
- And finally ACO-based classifier [14].

We also compare the performance of proposed classifiers with two simple and well-known classifiers, Naïve Bayes and TAN.

Furthermore, in order to construct more efficient networks, another scoring function is used, which is proposed in [31] and is called classification rate:

$$CR = \frac{1}{|D|}\sum_{m\in D}^{|D|}\delta(B_D\left(x_{1:N}^m\right), c^m) \tag{21}$$

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

Table 4 represents the average error rate of different algorithms for classification of different datasets. For each algorithm, we have carried out 10 independent runs over each dataset with a fixed execution time. The best results are highlighted.

As a reference for the goodness of the results, we can consider the average error rate for all 25 datasets, which are **0.120174** for BNC-VLA, 0.159136 for FALA, 0.161916 for ACO, 0.172716 for Genetic, 0.171472 for Hill Climbing, 0.18094 for NB, and 0.1586 for TAN. Moreover, for 20 datasets BNC-VLA has shown the best results. For five remaining datasets, Cleve, Crx, Flare, Sat image, and Vehicle, results are approximately the same as the best. The best results for Cleve and Crx are achieved by ACO and HC, respectively; this is justifiable by considering the stochastic nature of the algorithms. For other datasets TAN is superior to BNC-VLA. The volumes of data in Flare, Sat image, and Vehicle are large enough and moreover; the numbers of classes are small considering the number of attributes.

Therefore, since TAN constructs the network based on the conditional mutual information test, it has shown good results on them. However, the differences between the best results and the results achieved by BNC-VLA are negligible.

Figure 4 represents the scatter charts which directly compare the BNC-VLA classifier with other classifiers; points above the line y=x represent wherever the BNC-VLA has shown better performance in comparison with other algorithms. Figure 5 shows a bar chart which compares average classification error of all algorithms on different datasets.

All in all, the BNC-VLA has outperformed simple structures like NB and TAN; and in comparison with FALA-based algorithm, Genetic algorithm, ACO-based algorithm, and Hill climbing algorithm, again BNC-VLA has shown better results.

6.2.3 Comparison the computation time for classification

Finally, classification execution time of BNC-VLA is compared against other algorithms. All algorithms are

implemented and executed in. Net framework in a PC which has a single CPU of Intel(R) Core™2 Duo 3.33GHz and a 1GB memory. Table 5 shows the results after 10 independent runs for seven chosen dataset. Chosen datasets for these experiments are: Letter, Chess, German, Hepatitis, Breast, Glass2, and Iris. The results indicate that BNC-VLA needs less time for classification, in comparison with other algorithms.

7 Conclusion

In this paper a learning automata-based algorithm, named BNC-VLA, is proposed for BN structure training. Unlike the other learning automata-based methods which assign learning automata to the possible edges, in our algorithm each automaton is assigned to a random variable. Therefore, the number of learning automata reduces and the convergence speeds up. Furthermore, despite other algorithms, there is no additional phase to remove cycles from the constructed BN; it is simply done during the network construction by using variable-action set learning automata. Moreover two improvements are proposed, which can increase the quality of the constructed network and decrease the execution time. The time complexity of the proposed algorithm is $O(n^2)$. The convergence of BNC-VLA is theoretically proved. Convergence results confirm that by a proper choice of the learning rate, the probability of choosing a BN with the maximum score converges to one. Reported experimental results show that BNC-VLA is superior to other related algorithms based on the quality and performance measures. BNC-VLA has two main features: (1) it builds acceptable networks, and (2) it has lesser computational cost for network construction; these may be due to the learning capability and a negligible computational cost of learning automata. In comparison with two other learning automata-based methods, BNC-VLA can locate better structure, and also it exhibits superior speed of convergence. Moreover, since classification is one of the important applications of BNs which is used in varied fields, we examine the classification accuracy of the proposed algorithm. To do this, classification error is measured for different BNs on 25 different datasets. Again experimental results show that BNC-VLA classifier performs better than other classifiers.

References

1. Larrañaga P, Poza M, Yurramendi Y, Murga RH, Kuijpers CMH (1996) Structure learning of Bayesian networks by genetic algorithms: A performance analysis of control parameters. IEEE Trans Pattern Anal Mach Intell 18(9):912–926
2. Chickering DM, Geiger D, Heckerman D (1994) Learning Bayesian Network is NP-hard, Technical Report MSR-TR-94-14
3. Cheng J, Bell DA, Liu W (1997) An algorithm for Bayesian belief network construction from data. In: proceedings of AI & STAT'97, pp 83–90
4. Chow C, Liu C (1968) Approximating discrete probability distributions with dependence trees. IEEE Trans Inf Theory 14(3):462–467
5. Heckerman D, Geiger D, Chickering DM (1995) Learning Bayesian Networks: The Combination of Knowledge and Statistical Data, Technical Report MSR-TR-94-09
6. De C, Cassio P, Ji Q (2011) Efficient structure learning of Bayesian networks using constraints. J Mach Learn Res 12:663–689
7. Friedman N, Iftach N, Dana P (1999) Learning bayesian network structure from massive datasets: the sparse candidate algorithm. In: Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., pp 206-215
8. Larranaga P, Kuijpers CMH, Murga RH, Yurramendi Y (1996) Learning Bayesian Network Structures by Searching for the Best Ordering with Genetic Algorithms. IEEE Trans. Syst. Man Cybern. 26:487–493
9. Myers JW, Laskey KB, Levitt TS (1999) Learning Bayesian Networks from Incomplete Data with Stochastic Search Algorithms, Proceeding of UAI Conference, 476–485
10. Rezvani, Nabi A, Mohammad RM (2009) A Learning Automata-Based Technique for Training Bayesian Networks. In: International Conference on Advanced Computer Theory and Engineering (ICACTE 2009). ASME Press
11. Moradabadi B, Beigy H (2014) A new real-coded Bayesian optimization algorithm based on a team of learning automata for continuous optimization. Genet Program Evolvable Mach 15(2):169–193
12. Tsamardinos I, Brown LE, Aliferis CF (2006) The max-min hill-climbing Bayesian network structure learning algorithm. Mach Learn 65(1):31–78
13. Yuan C, Malone B, Xiaojian W (2011) Learning optimal Bayesian networks using A* search. In: IJCAI Proceedings-International Joint Conference on Artificial Intelligence, vol 22, p 2186
14. De C, Luis M, Fernandez-Luna JM, Gámez JA, Puerta JM (2002) Ant colony optimization for learning Bayesian networks. Int J Approx Reason 31(3):291–311
15. Hesar AS (2013) Structure Learning of Bayesian Belief Networks Using Simulated Annealing Algorithm. Middle-East J Sci Res 18(9):1343–1348
16. Murphy K (2001) An introduction to graphical models. Rap. tech
17. Pelikan M, David EG (2002) Bayesian optimization algorithm: From single level to hierarchy. University of Illinois at Urbana-Champaign, Champaign, IL
18. Gallagher M, Wood I, Keith J, Sofronov G (2007) Bayesian inference in estimation of distribution algorithms. In: Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, pp. 127–133. IEEE
19. Heckerman D (2008) A tutorial on learning with Bayesian networks. In: Innovations in Bayesian Networks. Springer, Berlin, pp 33-82
20. Thathachar MAL, Harita BR (1987) Learning automata with changing number of actions. IEEE Trans Syst Man Cybern SMG17:1095–1100
21. Narendra KS, Thathachar MAL (2012) Learning automata: an introduction. Courier Corporation

22. Thathachar MAL, Sastry PS (1985) A Class of Rapidly Converging Algorithms for Learning Automata. IEEE Trans Syst Man Cybern SMC-15:168–175
23. Lakshmivarahan S, Thathachar MAL (1976) Bounds on the convergence probabilities of learning automata. IEEE Trans Syst Man Cybern SMC-6:756–763
24. Beigy H, Meybodi MR (2006) Utilizing distributed learning automata to solve stochastic shortest path problems. Int J Uncertainty Fuzziness Knowledge Based Syst 14:591–615
25. Beinlich IA, Suermondt HJ, Martin Chavez R, Cooper GF (1989) The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. Springer, Berlin
26. Herskovits E (1991) Computer-based probabilistic-network construction, PhD diss., Stanford University
27. Lauritzen SL, Spiegelhalter DJ (1988) Local computations with probabilities on graphical structures and their application to expert systems. J R Stat Soc Ser B Methodol:157–224
28. Netica Netica Bayesian network software from Norsys. http://www.norsys.com
29. Feng G, Zhang J-D, Liao SS (2014) A novel method for combining Bayesian networks, theoretical analysis, and its applications. Pattern Recogn 47(5):2057–2069
30. Murphy PM, Aha DW (1995) UCI Repository of Machine Learning Databases, Available: http://www.ics.uci.edu/~mlearn/MLRepository.html
31. Kohavi R, John G (1997) Wrappers for Feature Subset Selection, Elsevier Science. Artif Intell 97:273–324
32. Pernkopf F (2005) Bayesian network classifiers versus selective k-NN classifier. Pattern Recogn 38(1):1–10

S. Gheisari received the B.S. degree in Computer Engineering from the Islamic Azad University of Shiraz in Iran in 2006. She received the M.S. degree in Computer Network Engineering from Islamic Azad University of Qazvin in Iran in 2009. Currently, she is a PhD student in technology Department at the Science and Research University, Tehran, Iran. Her research interests include Computer networks, Cognitive networks, learning systems, Bayesian networks

M. R. Meybodi received the B.S. and M.S. degrees in Economics from the Shahid Beheshti University in Iran, in 1973 and 1977, respectively. He also received the M.S. and Ph.D. degrees from the Oklahoma University, USA, in 1980 and 1983, respectively, in Computer Science. Currently he is a Full Professor in Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran. Prior to current position, he worked from 1983 to 1985 as an Assistant Professor at the Western Michigan University, and from 1985 to 1991 as an Associate Professor at the Ohio University, USA. His research interests include, channel management in cellular networks, learning systems, parallel algorithms, soft computing and software development.

M. Dehghan Takht Fooladi received his BSc in Computer engineering from Iran University of Science and Technology (IUST), Tehran, Iran in 1992,and his MSc and PhD from Amirkabir University of Technology (AUT), Tehran, Iran in 1995, and 2001, respectively. Since 1995, he has been a research scientist at Iran Telecommunication Research Center (ITRC) working in the area of Quality of Service provisioning and Network Management. He joined Computer Engineering Department of Amirkabir University of Technology in 2004.

M. M. Ebadzadeh received the B.Sc in Electrical Engineering from Sharif University of Technology, Tehran, Iran in 1991 and M.Sc in Machine Intelligence and Robotic from Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran in 1995 and his Ph.D. in Machine Intelligence and Robotic from Tlcom ParisTech, Paris, France in 2004. Currently, he is an Associate Professor in Computer Engineering Department, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran. His research interests include evolutionary algorithms, fuzzy systems, neural networks, artificial immune systems and artificial muscles.