Applied Soft Computing 11 (2011) 3373–3384

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A hybrid method for learning Bayesian networks based on ant colony optimization

Junzhong Ji*, Renbing Hu, Hongxun Zhang, Chunnian Liu

College of Computer Science and Technology, Beijing University of Technology, Beijing Municipal Key Laboratory of Multimedia and Intelligent Software Technology, Beijing 100124, China

ARTICLE INFO

Article history:
Received 7 June 2009
Received in revised form 23 August 2010
Accepted 3 January 2011
Available online 12 January 2011

Keywords:
Bayesian networks
Ant colony optimization
Variable search space
Heuristic
Function
Simulated annealing strategy

ABSTRACT

As a powerful formalism, Bayesian networks play an increasingly important role in the Uncertainty Field. This paper proposes a hybrid method to discover the knowledge represented in Bayesian networks. The hybrid method combines dependency analysis, ant colony optimization (ACO), and the simulated annealing strategy. Firstly, the new method uses order-0 independence tests with a self-adjusting threshold value to reduce the size of the search space, so that the search process takes less time to find the near-optimal solution. Secondly, better Bayesian network models are generated by using an improved ACO algorithm, where a new heuristic function is introduced to further enhance the search effectiveness and efficiency. Finally, an optimization scheme based on simulated annealing is employed to improve the optimization efficiency in the stochastic search process of ants. In a number of experiments and comparisons, the hybrid method outperforms the original ACO-B which uses ACO and some other network learning algorithms.

1. Introduction

Bayesian networks (BNs) are important probabilistic models within the field of artificial intelligence, and also powerful formalisms to model the uncertainty in the real world. A Bayesian network uses a graphical model to depict conditional independence among random variables in the domain and encodes the joint probability distribution. Given a network and observations of some variables, the values of other unobserved variables can be predicted by a probabilistic inference. Nowadays, many systems have been constructed based on this paradigm in a variety of different areas including vision recognition, medial diagnosis, trouble-shooting, information retrieval and so on.

With the development and popularity of BNs, earning BN structure from data has received considerable attention, and researchers have proposed various learning algorithms [1–13]. Generally, these algorithms can be classified into two main categories [3]: the dependency analysis approach, and the score-and-search approach. The first poses BN learning as a constraint satisfaction problem, and constructs a BN by dependency tests [2,3]. The second poses BN learning as an optimization problem, and uses a search method to find a network structure with the best score where a scoring metric is employed to evaluate candidate networks [1,4]. Unfortunately, both approaches have their own drawbacks. For example, the first approach has to perform an exponential number of dependency tests and some test results of higher order are unreliable, while the second approach often traps in a local optimum due to huge search spaces and the limitation of search methods. To solve these problems, new algorithms have been developed in recent years. For instance, there are three efficient approaches using a meta-heuristic mechanism to get the global near-optimum in the candidate network space. The first uses Genetic Algorithm (GA) [5,7], the second applies Evolutionary Programming (EP) [8,11], and the third employs ant colony optimization (ACO) [6,9]. Moreover, there is a research focus [10,11] that combines basic ideas of the dependency analysis approach and the score-and-search approach. These hybrid methods first use a dependency analysis method to reduce the search space of candidate solutions, then employ a score-and-search method to search in the reduced space. Different methods in dependency analysis and score-and-search phases can be used, which compose different hybrid methods.

In this paper, we propose a hybrid method to learn BNs. The hybrid method consists of two phases, namely, the Conditional Independence (CI) test phase and the search phase. In the CI test phase, order-0 independence tests with a self-adjusting threshold value are conducted to dynamically restrict search spaces of feasible solutions, so that the search process in the next phase can be accelerated while keeping good solution quality. In the search phase, an improved ACO for learning BNs is used to find good models. Here we use two techniques: 1. A new heuristic function

* Corresponding author.
E-mail address: jjz01@bjut.edu.cn (J. Ji).

1568-4946/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2011.01.009

combining the global score-increase of a solution with local mutual information between nodes is introduced to enhance the search effectiveness and efficiency. 2. An optimization strategy based on a Metropolis rule of simulated annealing is employed to further improve the optimization efficiency in the stochastic searching of ants. We call our new method hybrid ant colony optimization for Bayesian network learning (HACO-B). In a number of experiments, we perform an analytical study to compare the new method to ACO-B and some other network learning algorithms. The experimental results on benchmark data sets show that the hybrid algorithm outperforms the original ACO-B and some other network learning algorithms.

The paper is organized as follows. In Section 2, we present the background of Bayesian networks and the basic idea of the ant colony optimization for learning Bayesian networks. In Section 3, we describe our new algorithm in detail. Section 4 reports our experimental results. Finally, we conclude the paper in Section 5.

2. Background

2.1. Bayesian networks

A Bayesian network is a Directed Acyclic Graph (DAG) $G = \langle X,A\rangle$, where each node $X_i \in X$ represents a random variable in a domain, and each arc $a_{ij} \in A$ describes a direct dependence relationship between two variables X_i and X_j. Associated with each node X_i, is a conditional probability distribution represented by $\theta_i = P(X_i \mid \prod(X_i))$, which quantifies how much the node X_i depends on its parents $\prod(X_i)$. As the graph structure G qualitatively characterizes the independence relationship among random variables, the conditional probability distribution quantifies the strength of dependencies between a node and its parent nodes. It can be proved that a Bayesian network $\langle X,A\rangle$ uniquely encodes the joint probability distribution of the domain variables $X = \{X_1, X_2, \ldots, X_n\}$:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\Pi(X_i)) \quad (1)$$

2.2. Learning Bayesian network structures

The structure of a BN reflects the underlying probabilistic dependence relations among the nodes (corresponding perhaps to a database attribute) and a set of assertions about conditional independencies. The problem of learning a BN structure can be stated as follows: given a sample data $D = \{X[1], X[2], \ldots, X[N]\}$ where $X[i]$ is an instance of domain variables, the learning goal is to find the BN structure that best matches D. During the past decade, people have proposed many algorithms on learning Bayesian network structure. As mentioned above, there are two basic mechanisms. The first is an approach based on the dependency analysis [2,3], which takes the learning process as a constraint satisfaction problem, and then constructs a network structure by testing the conditional independence relations. The second is score-and-search approach [1,4], which takes the learning problem as a structure optimization problem. The latter uses a score metric to evaluate every candidate network structure, and then finds a network structure with the best score. Though the implementation of the former approach is relatively simple, the computations for high-order tests are complex and unreliable. Moreover, the precision of the learned model from the dependency analysis approach is hard to ensure, thus the score-and-search approach is gradually becoming a popular approach for learning Bayesian networks.

Given a node ordering, the parent nodes of each node in a BN, $\prod(X_i) = \{X_k : k \in \Phi(i)\}$, are only selected from the set of nodes preceding the current node X_i, namely, $\Phi(i) \subseteq \{1, 2, \ldots, i-1\}$, thus the number of possible parent sets is 2^{i-1} for each node X_i. Further, the number of possible structures for a BN with n nodes is $2^{n(n-1)/2}$ when a node ordering is known, and the complexity of a BN structure space is $n!\ 2^{n(n-1)/2}$ for the case of an unknown node ordering. Obviously, it is intractable for the complete search based on a score to find the global optimal solution when n is large. In the last few years, researchers proposed some effective algorithms [4,10,12] assuming a complete node ordering. Unfortunately, these algorithms still perform complete searching in the worst case, and they are unfit to learn a BN structure without a complete node ordering.

Though some improved hill-climbing algorithms [12,13] can also solve the problem of learning a BN structure with an unknown node ordering, they usually get a local optimal solution of the model. Recently, the development of stochastic search technologies has provided an effective and feasible method to tackle the problem. Genetic algorithms [5,7], evolutionary programming [8,11] and ant colony optimization [6,9] have been applied to learning Bayesian networks, respectively. These methods perform stochastically iterative searches and find the global best solution by means of simulating various natural phenomena.

2.3. Learning Bayesian networks using ACO (ACO-B)

2.3.1. Ant colony optimization

Ant colony optimization (ACO) is a meta-heuristic search algorithm, which was first proposed by Dorigo et al. in the 1990s [14,15]. Since then ACO has attracted a large number of researchers. As the theoretical framework of ACO has grown up in recent years [16–18], ACO is becoming popular, and it often gives satisfactory results for various optimization problems in a wide range of domains [19–21], such as data mining, machine learning, bioinformatics and multiple objective optimization problems. In addition, ACO plays a more and more important role in combination with other meta-heuristic mechanisms to effectively solve many NP-complete problems [22,23].

Initially, ACO was inspired by the observation of real ants looking for food. Ethnologists observed that ants can find the shortest path from their nest to the feeding food source by exploring and exploiting pheromone information, which has been deposited on the path when they traversed. They then can choose routes based on the amount of pheromone. Namely, ants communicate information about food source via pheromone, which they secrete as they move along. The larger amount of pheromone is deposited on a route, the greater is the probability of selecting the route by ants. Thus, when one ant finds a good short path from the nest to a food source, other ants are more likely to follow this path, and such a self-strengthening behavior eventually leads all the ants following the shortest path. The idea of the ACO is to mimic this behavior with artificial ants walking around the graph representing the problem to solve. While constructing the solutions, each artificial ant finds a solution starting from a start node and moving to feasible neighbor nodes step-by-step. During the process, the pheromone also evaporates over time, so that pheromone trails of infrequently traveled paths become weaker while frequently traveled paths are reinforced. Moreover, artificial ants not only imitate the learning behavior described above, but also employ problem-specific heuristic information to govern them to search towards neighbor nodes stochastically. Based on this mechanism, an effective ACO algorithm with the *K*2 metric for learning Bayesian networks, called ACO-B, is proposed in [6].

2.3.2. K2 metric

The *K*2 metric is a well-known evaluation measure for learning Bayesian networks from data, which uses a Bayesian scoring metric to measure the joint probability of a BN. The scoring metric is

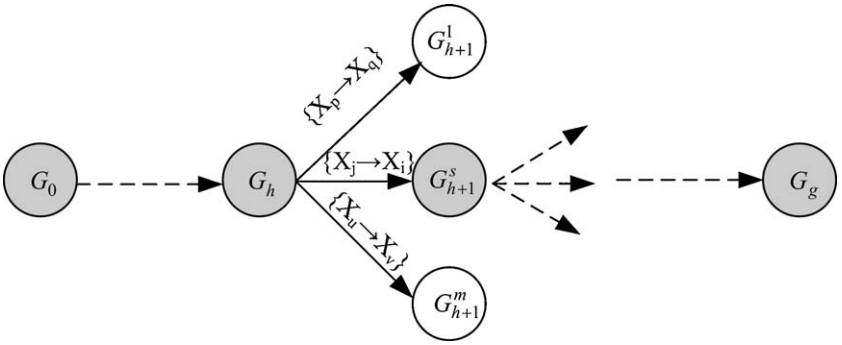

Fig. 1. The construction process of a BN by an ant.

referred as the *K*2 metric as it is used in the *K*2 algorithm [24]. The initial expression of the *K*2 metric is:

$$P(G, D) = P(G) \cdot \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{2}$$

where D is a given training set, G is a possible network structure, r_i is the number of possible values of the variable X_i, q_i is the number of possible configurations (instantiations) for the variables in $\prod(X_i)$, and N_{ijk} is the number of cases in D where X_i has its kth value and $\prod(X_i)$ is instantiated to its jth value.

Assuming an uniform prior for $P(G)$ and using $\log(P(G,D))$ instead of $P(G,D)$, the scoring metric $f(G{:}D)$, which evaluates G with respect to D, can be decomposed in the following way [6]:

$$f(G : D) = \sum_{i=1}^{n} f(X_i, \Pi(X_i)) \tag{3}$$

More formally, the function $f(X_i, \prod(X_i))$ is defined as:

$$f(X_i, \Pi(X_i)) = \sum_{j=1}^{q_i} \left(\log \left(\frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \right) + \sum_{k=1}^{r_i} \log(N_{ijk}!) \right) \tag{4}$$

As the joint probability is less than 1, the *K*2 metric using $\log(P(G,D))$ is always a negative value. In essence, the *K*2 metric is a Bayesian scoring metric in forms of either $P(G,D)$ or $\log(P(G,D))$. Thus, the best *K*2 value is the biggest one which is related to the optimal BN structure.

2.3.3. ACO-B algorithm

The ACO-B algorithm [6] is a score-and-search approach based on the ant colony optimization for learning Bayesian networks, whose main idea is to use the *K*2 scoring metric to evaluate a BN structure, and guide ants to search for the global maximum in a feasible solution space.

Let a be the number of ants in an ant colony, $\tau_{ij}(t)$ be the pheromone intensity associated with the directed arc a_{ij} at t time, and the initial pheromone intensity of every directed arc be a constant value C, i.e., $\tau_{ij}(0) = C$. A pheromone is a chemical substance released into the environment that can reflect communication between ants. In constructing a solution, each ant $k (k = 1, 2, \ldots, a)$ starts from the empty graph G_0 (arcs-less DAG) and proceeds by adding an arc at a time. The construction process of a BN for an ant is shown in Fig. 1, where the current state G_h of the ant is a graph with all nodes $X_i \in X$, exactly h arcs and no directed cycle, and there are m candidate directed arcs. If based on the pheromone and heuristic information of candidate arcs, the ant selects the sth arc a_{ij} as a new component of a solution, and the new state after adding an arc a_{ij} can be denoted as $G^s_{h+1} = G_h \cup \{a_{ij}\}$. Once there is no way to make the score of a BN structure higher by adding an arc, the construction process is ended and the ant gets its solution G_g.

The detailed process of constructing a solution can be described as follows. At time t, the probabilistic transition rule that an ant k selects a directed arc a_{ij} from the current candidate arcs is defined as

$$i, j = \begin{cases} \arg\max_{r,l \in DA_k(t)} \{[\tau_{rl}(t)] \cdot [\eta_{rl}(t)]^{\beta}\}, & \text{if } q \le q_0 \\ I, J, & \text{otherwise} \end{cases} \tag{5}$$

where $\tau_{rl}(t)$ and $\eta_{rl}(t)$ respectively represent the pheromone intensity and the heuristic information of the directed arc a_{rl}, the methods for calculating $\tau_{rl}(t)$ and $\eta_{rl}(t)$ will be discussed in the following; β is the weighted coefficient which controls $\eta_{rl}(t)$ to influence the selection of arcs; $DA_k(t)$ $(r,l \in DA_k(t))$ is the set of all candidate arcs that satisfy constraint conditions and whose heuristic information is larger than zero; q_0 $(0 \le q_0 < 1)$ is an initial parameter that determines the relative importance of exploitation versus exploration; q is a random number uniformly distributed in [0,1]; I and J are a pair of nodes randomly selected according to the probabilities in Eq. (5), with $\alpha = 1$.

$$p^k_{ij}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{r, \in DA_k(t)} [\tau_{rl}(t)]^{\alpha} \cdot [\eta_{rl}]^{\beta}}, & \text{if } i, j \in DA_k(t) \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where parameter α depicts the relative importance of the pheromone $\tau_{rl}(t)$ left by the real ants. As the learning goal is to achieve the best BN structure whose *K*2 score is the maximum, the heuristic information function of a directed arc can be interpreted as the greatest increase produced in *K*2 score when the arc is added to the graph. By means of decomposability of *K*2, the heuristic information function can be defined as

$$\eta_{ij}(t) = f(X_i, \Pi(X_i) \cup X_j) - f(X_i, \Pi(X_i)). \tag{7}$$

After each iteration of the ant colony is performed, the ACO-B algorithm will carry out the pheromone updating process, which includes local and global updating steps. First, while building a solution, if an ant selects an arc a_{ij}, then the pheromone level of the corresponding arc is changed in Eq. (7).

$$\tau_{ij}(t + 1) = (1 - \psi)\tau_{ij}(t) + \psi\tau_0 \tag{8}$$

where τ_0 is the initial pheromone level, and $0 < \psi \le 1$ is a parameter that controls the pheromone evaporation. Then, the algorithm finds the best solution from all feasible solutions obtained so far by means of the *K*2 metric, and performs the global updating for each arc of the current best solution. The global updating rule is

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\Delta\tau_{ij},$$

Algorithm: ACO-B

1. Initialization:

Initialize **a**, NC, G(0), ρ, φ, q_0, $G^+ = G(0)$,τ_0 and l_{step};

2. Loop:

For l=1 to NC do:

1) For k=1 to **a** do:

$G_k = AntConstructGraph()$;

If (l mod l_{step}=0) then $G_k = Optimization(G_k)$;

2) $G^+_{(l)} = argmax_{k:1\ldots\mathbf{a}} f(G_k : D)$;

3) If $(f(G^+_{(l)} : D) \geq f(G^+ : D))$ then $G^+ = G^+_{(l)}$;

4) Perform global pheromone updating by Eq.(8);

3. Local optimization:

1) For k=1 to **a** do: $G^o_k = Optimization(G_k)$

2) $G^+_{(l)} = argmax_{k:1\ldots\mathbf{a}} f(G^o_k : D)$;

3) If $(f(G^+_{(l)} : D) \geq f(G^+ : D))$ then $G^+ = G^+_{(l)}$;

4. Return G^+;

Fig. 2. Description of the ACO-B learning algorithm.

$$\Delta\tau_{ij} = \begin{cases} \dfrac{1}{|f(G^+ : D)|} & \text{if } a_{ij} \in G^+ \\ \tau_{ij} & \text{otherwise} \end{cases} \tag{9}$$

where $0 < \rho \leq 1$ is also a parameter of the pheromone evaporation, and $f(G^+{:}D)$ is the metric value of the best solution G^+.

Assuming that the ACO-B algorithm will end after running NC times, the global best solution is $G^+ = arg\ \max f(G^+_{(l)} : D)$, where $l \in (1,2,\ldots,NC)$ is the number of iterations, and $G^+_{(l)} = arg\ \max f(G_k : D)$ $(k \in 1, 2,\ldots,a)$ is the best solution at the lth iteration. Fig. 2 displays the overall process of ACO-B [6].

In Fig. 2, each ant uses the function called *AntConstructGraph*() to construct its solution and perform local pheromone updating. To improve the quality of a solution, the ACO-B algorithm advisably employs the *Optimization*(.) function, which uses the standard operators of addition, deletion and reversal of arcs, to locally optimize the obtained solution G_k.

Since ACO-B algorithm adopts the stochastic search mechanism based on ant colony optimization, it can get the global near-optimal solution, and the quality of solutions is higher than that of many deterministic search methods [6]. However, there is a drawback of ACO-B algorithm, namely, the convergence time is too long. The main reason for this is that even if ACO-B does not traverse the whole candidate solution space during each iteration, ants may still select some candidate arcs that cannot be components of the best solution at all. In other words, ACO-B might consider many useless combinations wasting much running time. This inspired us to present a hybrid algorithm, which integrates conditional independence tests, an improved ACO-B and a simulated annealing optimization strategy.

3. HACO-B algorithm

The efficiency of ACO-B can be enhanced by employing a number of strategies. First, order-0 independence tests with lower cost are performed to discover some potential constraints (i.e. independence knowledge) from the sample data D, and the search space is effectively reduced by using the obtained knowledge. Second, the obtained knowledge is reused to revise the heuristic function of ACO and guides ants to carry out fast searching in the reduced space. Third, a simulated annealing strategy is employed to control the optimization process, and further improves the convergence performance.

Since this hybrid method and an ant colony optimization algorithm are used in the Bayesian network learning, we call this method HACO-B. In the following sections, these ideas will be discussed in detail.

3.1. Order-0 independence tests with adaptable cutoff values

Based on the completely connected graph, ants in ACO-B construct their feasible solutions from G_0 (arcs-less DAG) by adding a directed arc to the current graph each time. Each ant could select a satisfied arc from the candidate connect graph at every iteration, thus the complexity of the initial candidate connect graph determines the complexity of ACO-B algorithm to a large extent. If we adopt some strategies to make the initial connect graph simplified, the search space of the algorithm will be greatly reduced. In light of the idea of the constraint satisfaction, HACO-B algorithm first introduces Conditional Independence (CI) tests [10] to reduce the search space before ants perform searching.

For the BN structure learning, the CI test is a typical method that validates the conditional independence relationship between two variables given a conditional set. The basic concept of CI tests is the measure of an information flow in information theory. A simple and natural measure for the information flow between X_i and X_j is the mutual information:

$$Inf(X_i, X_j|Z) = \sum_{x_i,x_j,z} \hat{P}(x_i, x_j, z)\ \log \frac{\hat{P}(x_i, x_j|z)}{\hat{P}(x_i|z)\hat{P}(x_j|z)} \tag{10}$$

where Z is a given condition set, $\hat{P}$ denotes a probability estimate for various cases in the sampling data set D, and x_i, x_j, and z respectively correspond to the observed values of the variables and the condition variable set. Let $t = 2 \times N \times \mathrm{Inf}(X_i, X_j|Z)$. If N is large enough, then t obeys χ^2 distribution [2]. Therefore, by means of comparing t with a cutoff threshold value γ, we can conduct χ^2 tests and determine the indirectly connected relationships between two variables. In effect, if $t > \gamma$, then the directly connected relationship is maintained. Otherwise, it is deleted.

Considering the reliability and less computational cost of the low order CI tests, the HACO-B algorithm only adopts order-0 independence tests (Z is a null set) in the CI test phase. More specifically, we first build an undirected complete graph including all variables, and then compute the mutual information $Inf(X_i, X_j)$ for each arc of the completely connected graph. By means of χ^2 test, the algorithm confirms some undirected relationships between variables and computes the Forbidden Connect Set (*FCS*) using these relationships. Then, all redundant connect arcs in *FCS* are removed from the completely connected graph, which changes to a possible connect graph.

The more constraint knowledge obtained by CI tests, the smaller the search space of BNs. However, an improper cutoff value γ could bring detrimental results and influence the quality of solutions. For instance, if some crucial edges (which must appear in the best solution) fail to pass the CI tests, they will be excluded outside the set of candidate arcs, thus it is impossible for ants to obtain the best structure in the subsequent search process. Even if there is an arc addition operator in local optimization, the arcs accidentally deleted might not be added back to the the near-optimal solution. To tackle this problem, we adopt a new strategy where the cutoff value can be adapted to fit the iteration process. First, we set a larger initial $\gamma = \max(t)/300$, then let the value of γ decrease as the number of iterations increases. That is, our algorithm conducts more rigid χ^2 tests at the beginning, allowing ants to quickly construct some feasible solutions from a smaller search space. However, the solutions obtained by the approach may be imperfect because the constraints are too strict to keep some important arcs from being deleted. As the value of γ decreases, the constraint conditions grad-

ually loosen and search spaces are enlarge correspondingly, so ants can select from many more candidate arcs, including some of the crucial arcs previously lost. The strategy can retain a good quality of solutions while the search space is still effectively reduced.

To illustrate the change of candidate connect graphs with the reduction of γ value, we take a network model with 4 nodes as an example, shown in Fig. 3. Given a larger γ at the beginning, we perform order-0 CI tests and get three conditional independency assertions, $I(X_1,X_3)$, $I(X_1,X_4)$, and $I(X_3,X_4)$. Using these constraint knowledge, we can delete some redundant arcs, and get a possible connect graph shown as Fig. 3(a). With the increase in the number of iterations, the value of γ will decrease, and the constraint condition will relax. If only two independency assertions $I(X_1,X_3)$ and $I(X_1,X_4)$ are obtained at certain iteration, then we get another possible connect graph shown as Fig. 3(b). When the value of γ approaches 0 after a number of iterations, then there is a direct arc between every two nodes and the completely connected graph shown in Fig. 3(c) will be obtained.

The different initial candidate connected graphs will directly influence the size of search spaces at different iterations. Corresponding to the connected graphs in Fig. 3, the changes of search spaces for the parent set of X_4 are given in Fig. 4. Since interrelated arcs failed to pass the CI tests, those candidate parent sets of X_4, shown shaded in Fig. 4 could be pruned. It is obvious that the more constraints, the smaller a search space is, e.g., Fig. 4(a) presents that a search space only has two candidate sets of parent nodes $\{\}$,$\{X_2\}$. Fig. 4(b) depicts that the search space extends the four candidate sets of parent nodes $\{\}$,$\{X_2\}$,$\{X_3\}$,$\{X_2,X_3\}$. And Fig. 4(c) shows that the search space equals to the whole state space in the worst case.

Since many network structures, including the arcs that failed to pass CI tests, will be prevented from being constructed, the search space is greatly reduced (especially in the early iterations). With the search space subsequently expanding, the negative effects of the inappropriate selection of γ will be eliminated. Thus the quality of solutions can be ensured.

3.2. Heuristic function with a weighted factor

In ACO-B, the heuristic function is defined as the score-increase introduced by the operator of an arc addition. According to the decomposability of *K2* metric, the operator that adds an arc $X_j \to X_i$ to the current G_h will bring the score change:

$$f(G_{h+1} : D) - f(G_h : D) = f(X_i, \Pi(X_i) \cup X_j) - f(X_i, \Pi(X_i)). \tag{11}$$

This heuristic function definition also reflects the score increase of the changing structure, thus representing the global information of the solution. However, the definition has a drawback. Namely, it only gives the heuristic information of arcs by the evaluation of an arc combination (solution structure), but does not consider each arc's own connecting intensity, implied in sample data D. This is apt to make the heuristic information unilateral. Therefore, we redefine the heuristic function of a directed arc:

$$\eta_{ij}(t) = \omega \cdot (f(X_i, \Pi(X_i) \cup X_j) - f(X_i, \Pi(X_i))) \tag{12}$$

where ω is a weighted factor concerned with the arc connecting intensity, and its value is larger than 1. In light of information theory, the mutual information $Inf(X_i,X_j)$ of X_i and X_j can objectively reflect whether the two nodes in BN are dependent and how much the dependency is. Intuitively, X_i and X_j are independent of each other when $Inf(X_i,X_j) = 0$, otherwise the bigger the value of the mutual information, the stronger the dependence between the two nodes. It shows that the value of the mutual information can also be used as heuristic knowledge to guide ants' activity of selecting arcs. Thus we define the weighted factor as $\omega = 1 + Inf(X_i, X_j)$, which employs the local dependency information (mutual information) of arcs to control ants' selection of arcs. Obviously, when the dependency intensity is stronger and the score-increase is larger, the heuristic information becomes greater, and vice versa. That is, the new definition integrates the global solution information with the local component information, which can give ants much better insight in the selection of arcs.

3.3. Optimization strategy based on simulated annealing

To overcome the local optimum and improve the quality of solutions, the ACO-B algorithm employs a local optimizer to perform greedy searches in some neighbors of the current solution. However, the original optimizing process is conducted in light of a fixed length of iterations. The study did not provide much attention to the efficiency of optimizing processes. In other words, the original method might waste some running time when the local optimizer is not required. Therefore, we introduce an optimization strategy based on a simulated annealing strategy to enhance optimization efficiency in the HACO-B algorithm. Before conducting the optimizing process at a certain step, HACO-B compares the best solution in the current iteration with that of last iteration, and then determines whether to carry out the local optimizing process. The practical Metropolis rule can be denoted as:

$$P = \begin{cases} 1, & \text{if } \Delta F \le 0 \\ \exp(-\Delta F / t_l), & \text{otherwise} \end{cases} \tag{13}$$

where t_l is the annealing temperature $t_l = \lambda \cdot t_{l-1}$ ($l = 2, 3, \ldots, NC$) and $\lambda < 1$ is a control parameter, and ΔF is the score difference of the solution obtained at two iterations. If the score of the solution at the current iteration is smaller than that of the last iteration, then the optimizing process is necessarily carried out. On the contrary, when the score of the solution at the current iteration is better, the optimizing process is randomly carried out at a certain probability. Moreover, the annealing temperature will reduce as the ACO runs, hence the random optimizing process will gradually decrease. When $t_l \to 0$, the strategy only performs optimization for the cases of stagnating solutions.

3.4. Algorithm description

Combined with the three aforementioned strategies, the algorithm of HACO-B is summarized in Fig. 5.

From Fig. 5, we can see that HACO-B is a hybrid algorithm which merges the dependence analysis approach with the score-and-search approach. First, HACO-B employs the adaptable cutoff value to dynamically change search spaces of different iterations. This not only effectively restricts search spaces, but also prevents the crucial arcs from being lost if the parameter is not appropriate. Secondly, the knowledge from CI tests is reused in the new heuristic function, which enhances the insight ability of the heuristic function. That is, the new algorithm not only makes use of the constraint knowledge to reduce search spaces, but also uses constraint knowledge as the heuristic information that guides searching. Finally, a simulated annealing strategy is used to control the local optimizing process, which can save time in the optimization process.

3.5. Algorithm analysis

Similar to all other stochastic search algorithms for learning BNs, the main cost of ACO-B is the computation of statistical factors. Each new search object needs to carry out new statistical counts, so each iteration of ants contributes much to the computing cost. And in the case of same sample capacity, the more the number of iterations, the more the computing cost. In contrast to ACO-B, HACO-B not only employs the constraint knowledge to reduce search spaces, but also takes the constraint knowledge as a kind of heuristic knowledge to guide the process of stochastic searches. Moreover, HACO-B

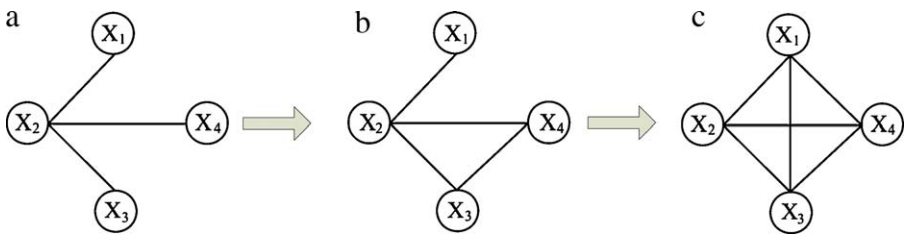

Fig. 3. The changes of candidate connected graphs with the reduction of γ value.

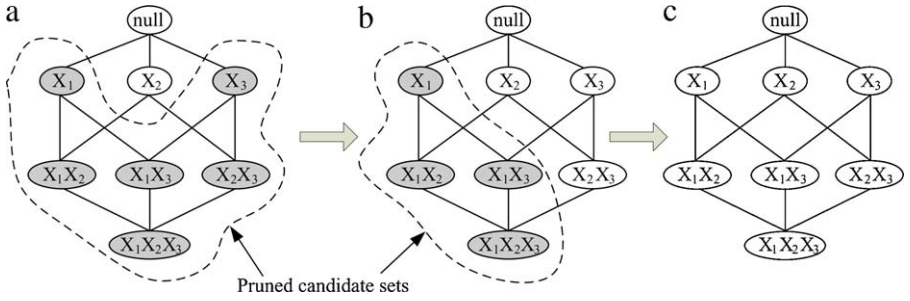

Fig. 4. The changes of search spaces for the parent set of X_4.

applies a simulated annealing strategy to improve the process of the local optimization. In essence, the employment of these three new strategies decreases the computing cost of stochastic searches and improves the time performance of the ant colony optimization.

Theoretically, the more the constraint knowledge obtained by CI tests, the smaller the search space and the higher the searching efficiency. However, the results of higher-order CI tests may be unreliable [2], and there is some extra computing cost even for lower-order CI tests. For instance, since the number of order-0 CI tests is C_n^2, the computing complexity is $O(n^2)$; The number of order-1 CI tests is $C_n^2 \cdot C_{n-2}^1$, hence the computing complexity is $O(n^3)$; Similarly, the number of order-2 CI tests is $C_n^2 \cdot C_{n-2}^2$, the computing complexity is $O(n^4)$. Therefore, HACO-B only uses order-0 CI tests to reduce the search space. Even so, the experimental results show that the pruning operator based on order-0 CI tests may accidentally delete a few candidate arcs in the best solution. Hence, HACO-B makes cutoff values vary with different iterations, and yields variable search spaces with different sizes. Though this costs extra time, the detriment caused by an inappropriate cutoff value is overcome.

The convergence is an important character of ant colony optimization, where the convergence of the ACS algorithm has been proved by a theorem in [18]. Let $P^*(t)$ be the probability that the algorithm finds an optimal solution at least once within the first t iterations. The theorem pointed that the $P^*(t)$ is asymptotically close to 1 for a sufficiently large t, and the condition that the theorem holds is only $0 < \eta_{ij} < +\infty$ for each pair (i,j) and $\beta < +\infty$. Both ACO-B and HACO-B are ACO algorithms in the ACS formalism, and the heuristic function of HACO-B is weighted for one of the ACO-

Algorithm: HACO-B
1. Initialization:
Initialize **a**, NC, G(0), l_{step}, ρ, φ, q_0, $\gamma(0)$, λ, t_0, $G^+ = G(0)$, and $\tau_{ij} = 1/n \cdot |f(G(0) : D)|$;
2. Condition Independent Test Phase:
For every pair of nodes $(X_i, X_j) \in X$ do:
Perform order-0 CI tests;
Store the t-value in the matrix P_v;
3. Search Phase by Ant Colony Optimization Algorithm:
For l=1 to NC do:
3.1 Obtain a initial search space for the current iteration:
$\gamma(l) = \gamma(l-1) - \Delta\gamma$;
$t_l = \lambda \cdot t_{l-1}$;
Refine the search space by checking the $\gamma(l)$ against the P_v;
For every pair of nodes $(X_i, X_j) \in FCS$ do:
$\eta_{ij} = -\infty, \eta_{ji} = -\infty$;
3.2 Construct a solution in the reduced search space:
For k=1 to **a** do:
$G_k = AntConstructGraph()$;
3.3 Perform the solution optimization based on the simulated annealing strategy:
If (l mod l_{step}=0) then
{ If ($F(G_{(l)}^+$:D)$\leq F(G_{(l-l_{step})}^+$:D) or random $\leq exp(-(F(G_{(l)}^+$:D)-$F(G_{(l-l_{step})}^+$:D))/t_l))
then
{ For k=1 to **a** do:
$G_k = Optimization(G_k)$ } };
$G_{(l)}^+ = argmax f(G_k : D)$;
If $(f(G_{(l)}^+ : D) \geq f(G^+ : D))$ then
$G^+ = G_{(l)}^+$;
3.4 Perform global pheromone updating by Eq.(9);
4. Return G^+;

Fig. 5. The HACO-B algorithm.

Table 1
Data sets used in our experiments.

Data set (D)	Original network (G)	Sizes of D	Nodes of G	Srcs of G	*K2* score of G
Alarm 1000	Alarm	1000	37	46	−5023.28
Alarm 2000	Alarm	2000	37	46	−9717.46
Alarm 3000	Alarm	3000	37	46	−14401.29
Alarm 4000	Alarm	4000	37	46	−19098.41
Alarm 5000	Alarm	5000	37	46	−23781.98
Alarm 6000	Alarm	6000	37	46	−28347.11
Alarm 7000	Alarm	7000	37	46	−33022.93
Alarm 8000	Alarm	8000	37	46	−37745.28
Alarm 9000	Alarm	9000	37	46	−42361.50
Alarm 10000	Alarm	10,000	37	46	−47076.20
Insurance 10000	Insurance	10,000	27	52	−57567.62
Asia 1000	Asia	1000	8	8	−9800.13

B. Since the *K2* score-increase of ACO-B is always larger than 0 for the candidate arcs and the weighted factor $\omega \geq 1$, both HACO-B and ACO-B satisfy the condition in the theorem, consequently ensuring the convergence of HACO-B.

One problem is that though local optimizations are very useful for getting a global optimum, these processes also cost more running time. HACO-B adopts the optimizing strategy based on simulated annealing to remedy this drawback. In HACO-B, the optimizing process is dynamically adjusted in light of the convergence of solutions. On the one hand, the local optimization is carried out when the evolution of solutions is stagnant. On the other hand, it is also called randomly when the evolvement is durative. Therefore, the strategy can not only save optimization time, but also keep up with the diversity of solutions.

4. Experimental evaluation

To assess the performance of the HACO-B algorithm, we use a common evaluation method, which is to test the algorithm on data sets generated from known networks using probabilistic logic samples. We test HACO-B on 12 different data sets, and compare the results with that of the original ACO-B and other new algorithms on the same data sets. All of the data sets are generated from well-known benchmarks of Bayesian networks including the Alarm, the Asia and the Insurance. Table 1 shows a summary of the data sets used in our experiments.

The experimental platform was a PC with Pentium 4, 2.8 GHz CPU, 512 M memory, and Windows XP. The algorithm was implemented by *Java*. By large numbers of experiments, the main parameters were set as follows: $\alpha = 1$, $\beta = 2$, $\rho = \psi = 0.4$, $q_0 = 0.8$, $NC = 100$, $t_0 = 3000$, $a = 10$, $l_{step} = 20$, $\Delta\gamma = 0.1$ and $\lambda = 0.99$. Here a, l_{step}, λ and $\Delta\gamma$ are parameters which may mainly influence performances of HACO-B. How to select their values will be emphatically discussed in the following subsection.

4.1. Performance analysis of HACO-B

We study the factors that affect the performance of HACO-B. Particularly, we wish to investigate the contributions of three strategies and the effects of different parameter selections on HACO-B performance.

4.1.1. Contributions of three strategies

We employed four algorithms to learn a BN structure from the Alarm data sets with different sample sizes. The four algorithms were called the original ACO-B, the ACO-B1 (only using variable search spaces), the ACO-B2 (only using the new heuristic function) and the ACO-B3 (only using the simulated annealing strategy). The experimental results are shown in Table 2, where the performance of algorithms is evaluated using three measures: *K2*, *It.* and *Time*. *K2* denotes the *K2* metric values of the solutions obtained for different sample capacities. *It.* is the number of iterations when the algorithm finds a near optimal structure. *Time* is the execution time (s) when the algorithm finds the near optimal structure. Results in the form $\mu \pm \sigma$ indicate the mean μ and the standard deviation σ over 10 executions independently carried out by the corresponding algorithm. Moreover, numbers in parentheses in the *K2* column are the best results found over 10 executions, and numbers in parentheses in the *It.* and *Time* columns are the smallest numbers of the iterations and the shortest running time when the best *K2* were obtained, respectively. The bold numbers are the best values in different cases (the same as in following tables).

By analyzing these data in Table 2, we can draw the conclusion that all three strategies introduced in the paper can evidently improve the running time of the ACO-B algorithm. More specifically, (1) the order-0 CI tests with variable cutoff values not only effectively enhance the time performance but also improve the solution quality on a majority of data sets. As the time saved during searching is much longer than the time increased in CI testing (especially when the sample size is large), the strategy can improve the time performance of ACO-B. Moreover, the logical reduction of search space also ensures the algorithm to obtain better solutions in limited iteration cycles. (2) The new heuristic function can improve the convergence performance (the number of the iterations and the running time) while keeping the solution quality. The main reason is that the improvement of the heuristic ability increases the diversity of solutions, thus ACO-B2 is apt to escape from a local optimum, and reduces the number of the iterations and the running time. (3) The new optimization strategy can improve the convergence performance, and the improvement of the running time is especially remarkable. This suggests that the optimization strategy based on solution convergence instances can effectively control the optimization process, saving optimization time.

It is obvious that the three aforementioned strategies, derived from three different phases, are all effective in the improvement of the performance of the ACO-B algorithm. This fact encourages us to put these strategies into a new algorithm (HACO-B) to get even better results.

4.2. Effects of different parameter selection

In this experiment, we test HACO-B with different parameter settings on the Alarm when the sample capacity is 5000. Table 3 summarizes the performance of HACO-B with 10 different ant colony sizes (a). The best and worst scores are respectively the highest and lowest scores of the 10 trails. The average score indicates the mean and the standard deviation of *K2* measurements over 10 trails, and the running time is the average running time for 10 trails.

A large ant colony means that more search points are employed, and as reflected by three *K2* scores, better solutions are obtained than in a smaller ant colony. However, the search time also increases proportionally to the size of the ant colony. To acquire

Table 2
The contributions of three strategies on ACO-B.

Sample capacity	ACO-B			ACO-B1			ACO-B2			ACO-B3		
	K2	It.	Time (s)	K2	It.	Time (s)	K2	It.	Time (s)	K2	It.	Time (s)
1000	−5024.14 ±0.34 (−**5023.28**)	75.20 ±4.61 (79)	54.58 ±2.10 (55.95)	−**5023.66** ±0.22 (−**5023.28**)	71.90 ±2.80 (60)	**32.39** ±0.89 (**28.50**)	−5024.16 ±0.58 (−**5023.28**)	**65.80** ±8.39 (**32**)	48.44 ±3.83 (33.89)	−5024.56 ±0.38 (−**5023.28**)	74.0 ±6.78 (56)	34.4 ±1.54 (31.32)
2000	−9717.64 ±0.11 (−**9717.46**)	59.30 ±6.96 (30)	95.71 ±5.76 (67.06)	−9717.79 ±0.24 (−**9717.46**)	50.0 ±5.37 (**20**)	**47.41** ±3.04 (**26.61**)	−**9717.57** ±0.08 (−**9717.46**)	50.30 ±7.45 (38)	84.05 ±5.77 (77.17)	−9718.25 ±0.59 (−**9717.46**)	**45.80** ±5.23 (40)	56.75 ±2.88 (54.01)
3000	−14402.01 ±0.37 (−**14401.29**)	72.10 ±6.32 (61)	196.93 ±9.35 (177.66)	−**14401.54** ±0.19 (−**14401.29**)	**60.80** ±7.32 (**40**)	**100.78** ±12.72 (**63.18**)	−14401.88 ±0.35 (−**14401.29**)	61.40 ±5.42 (40)	144.92 ±6.61 (117.77)	−14402.10 ±0.40 (−**14401.29**)	72.60 ±5.97 (44)	107.03 ±3.74 (93.07)
4000	−19099.64 ±0.65 (−**19098.41**)	66.70 ±6.07 (60)	247.79 ±10.97 (256.45)	−**19098.76** ±0.33 (−**19098.41**)	69.40 ±5.65 (60)	**110.68** ±17.27 (**62.04**)	−19099.26 ±0.69 (−**19098.41**)	**62.50** ±7.95 (**20**)	193.09 ±11.33 (122.81)	−19100.18 ±1.15 (−**19098.41**)	64.30 ±5.53 (43)	138.07 ±4.49 (114.37)
5000	−23782.17 ±0.13 (−**23781.98**)	72.30 ±4.78 (48)	272.58 ±10.67 (213.70)	−**23782.12** ±0.04 (−**23781.98**)	78.10 ±6.97 (80)	183.24 ±18.76 (184.67)	−23782.72 ±0.52 (−**23781.98**)	**66.50** ±5.06 (55)	248.16 ±8.06 (235.95)	−23782.55 ±0.33 (−**23781.98**)	69.50 ±4.90 (**51**)	**179.74** ±4.43 (**159.96**)
6000	−**28347.17** ±0.03 (−**28347.11**)	65.60 ±4.71 (40)	315.59 ±8.91 (265.58)	−28347.50 ±0.20 (−**28347.11**)	65.50 ±6.48 (60)	**153.20** ±24.91 (**106.98**)	−28347.67 ±0.34 (−**28347.11**)	**46.90** ±5.49 (**20**)	266.83 ±14.19 (201.53)	−28347.50 ±0.16 (−28347.11)	61.30 ±7.28 (63)	202.07 ±10.78 (215.56)

a balance between getting a better solution and using less time, we recommend an ant colony size of 10 ($a = 1$).

As described in Section 3.1, each search space has its own cutoff value γ. The next search space will inherit the cutoff value from its former one with a possible decrement by $\Delta\gamma$. Thus, each ant colony performs its search in a different search space and this search space can be modified dynamically by changing the cutoff value of CI tests. To investigate the effect of $\Delta\gamma$ on HACO-B, we perform experiments using different values of $\Delta\gamma$. The results are presented in Table 4.

From the table, we notice that the best score can be obtained for seven cases except $\Delta\gamma = 0.20$, the best result is obtained for $\Delta\gamma = 0.14$. However, it is not significantly different from the scores for other values of $\Delta\gamma$. Moreover, two smaller running times are achieved for $\Delta\gamma = 0.10$ and 0.09. They are significantly smaller than those of the other values of $\Delta\gamma$, which shows that a smaller search space can improve the time performance of HACO-B. Thus, we recommend a $\Delta\gamma$ value of 0.10 because the corresponding score values are comparable to that of $\Delta\gamma = 0.14$ and its running time is also close to the best result.

In our algorithm, we introduce a simulated annealing rule to control the local optimizing process. To investigate the effect of the simulated annealing strategy on HACO-B, we perform experiments using different values of λ. Table 5 gives the experimental results. We observe that there are some differences in running time for different values of λ. However, there is no significant difference among the three score values. To get the best result, we can select $\lambda = 0.85$. Of course, we can select $\lambda = 0.99$ if we especially pay attention to time.

Moreover, to study the effect of different optimization intervals on HACO-B, we perform experiments using different values of l_{step}. The experimental results are presented in Table 6. We notice that both running time and score value approximately decrease when the value of l_{step} increases. This shows that the more frequent the local optimizing process, the better the quality of solution obtained. However, it also costs much more time. Thus, we recommend $l_{step} = 20$ after considering effects on factors of running time and solution quality.

4.3. Comparing HACO-B with ACO-B

Table 7 provides a summary of the performance comparison between the HACO-B and ACO-B algorithms on five different data sets. Both algorithms are independently executed 10 times for each data set, and the figures are, therefore, an average of 10 trails.

In Table 7, *A.*, *D.* and *I.* are used to denote the structure differences between the learned and the original network, namely, the number of arcs accidentally added (*A.*), deleted (*D.*) and inverted (*I.*), compared with the original network. *Total Num.* represents the total number of statistics evaluated for the instances of all local structures during the learning process. *Practical Num.* represents the number of statistics truly computed from data. Since we have used the hashing techniques to cache the results, we avoid the necessity of recomputing previously calculated values, which can greatly save the running time. The meanings of other items and data formats are the same as those of Table 2.

Compared with ACO-B, HACO-B can always find better or equally good network structures for all the data sets in terms of both *K2*

Table 3
Comparisons of the *K2* score and running time for different *a*.

Ant colony sizes	Best score	Worst score	Average score	Running time (s)
2	−23782.06	−23892.97	−23806.43 ± 11.56	26.51 ± 1.67
4	−23782.15	−23866.97	−23792.51 ± 0.02	41.25 ± 4.08
6	−**23781.97**	−23786.71	−23783.10 ± 0.22	58.43 ± 2.66
8	−23782.06	−23783.24	−23782.21 ± 0.12	84.92 ± 3.04
10	−**23781.97**	−**23782.15**	−**23782.03** ± 0.02	98.75 ± 7.54
12	−**23781.97**	−23784.16	−23782.38 ± 0.23	114.06 ± 5.28
14	−**23781.97**	−23786.52	−23782.95 ± 0.48	137.17 ± 8.59
16	−**23781.97**	−**23782.15**	−23782.06 ± 0.02	144.88 ± 13.42
18	−**23781.97**	−**23782.15**	−23782.06 ± 0.02	158.55 ± 9.85
20	−**23781.97**	−23784.25	−23782.27 ± 0.22	173.17 ± 11.92

Table 4
Comparisons of the *K2* score and running time for different $\Delta\gamma$.

$\Delta\gamma$	Best score	Worst score	Average score	Running time (s)
0.33	−23781.97	−23784.25	−23782.29 ± 0.22	103.23 ± 7.05
0.25	−23781.97	−23782.15	−23782.05 ± 0.02	101.56 ± 6.19
0.20	−23782.06	−23782.15	−23782.08 ± 0.02	99.37 ± 8.68
0.17	−23781.97	−23782.15	−23782.05 ± 0.22	102.89 ± 6.32
0.14	−23781.97	−23782.06	−23782.03 ± 0.01	97.58 ± 8.84
0.12	−23781.97	−23782.15	−23782.02 ± 0.02	90.69 ± 4.65
0.10	−23781.97	−23782.15	−23782.05 ± 0.02	86.92 ± 6.60
0.09	−23781.97	−23784.25	−23782.27 ± 0.22	84.38 ± 4.44

Table 5
Comparisons of the *K2* score and running time for different λ.

λ	Best score	Worst score	Average score	Running time (s)
0.99	−23781.97	−23784.34	−23782.53 ± 0.29	91.34 ± 5.36
0.98	−23782.06	−23784.34	−23782.54 ± 0.25	97.48 ± 4.52
0.97	−23781.97	−23783.24	−23782.15 ± 0.12	107.63 ± 5.79
0.96	−23781.97	−23784.16	−23782.29 ± 0.21	100.94 ± 7.81
0.94	−23781.97	−23789.36	−23782.82 ± 0.72	107.01 ± 5.83
0.92	−23781.97	−23784.34	−23782.73 ± 0.33	102.94 ± 5.13
0.9	−23781.97	−23783.24	−23782.31 ± 0.16	113.83 ± 4.13
0.87	−23781.97	−23785.42	−23782.60 ± 0.38	102.89 ± 5.32
0.85	−23781.97	−23782.15	−23782.07 ± 0.02	102.63 ± 6.24
0.83	−23782.06	−23784.34	−23782.32 ± 0.23	100.02 ± 4.17
0.8	−23782.06	−23784.34	−23782.52 ± 0.29	98.71 ± 5.94

score and structure difference. On the other hand, the *Total Num.* and the *Practical Num.* are evidently reduced on all data sets, and the *It.* is also reduced on most of data sets, hence the *Time* of HACO-B is greatly improved.

Since the *K2* score is a key measure to evaluate the quality of solutions for different methods, we compare the *K2* scores of the two algorithms on ten Alarm data sets. Table 8 provides a summary of the performance comparison between two algorithms, where the best score is the optimum result in ten trails (numbers in the parentheses are the number of hits-runs out of 10), the average score is an average of ten trails, and the standard deviation is corresponding deviation amplitude. Apart from three cases (i.e. Alarm 5000, 6000 and 9000), HACO-B algorithm outperforms ACO-B algorithm on the quality of solutions.

In Fig. 6, we compare the iteration numbers of two algorithms on ten data sets with different sample capacities. For each algorithm, we record the iteration number averaged over 10 runs when obtaining the *K2* scores shown as Table 8 on the respective data sets. From Fig. 6, we observe the iteration number is not specifically related to the sample capacity, which illuminates the stochastic property of ACO. Furthermore, the new strategies adopted by HACO-B do not make the iteration process postponed. On the contrary, there are apparent improvements on the iteration number for half of the data sets.

Fig. 7 gives the time performance comparison between two algorithms, which corresponds to Fig. 6. We can see that HACO-B performs better than ACO-B in terms of the running time on all data sets. Moreover, the advantage in running time is most remarkable when the data set is large, namely, the bigger the sample size, the more obvious the improvement. The first reason for this is that HACO-B takes the learned mutual information as a heuristic knowledge to revise the heuristic function, which can enhance the insight of the heuristic function and improve the time performance of the ACO searching algorithm. The second reason is that HACO-B employs order-0 CI tests with self adjusting cutoff values to effectively reduce the search space, which can cut down much computation of statistical factors, scoring of structures and comparison of solutions. The last reason is that HACO-B uses the simulated annealing strategy based on the solution's evolution to control the local optimizing process, which can save optimizing time. The experimental results also show that the running time of ACO-B increases more quickly as the sample capacity increases. However, HACO-B is not sensitive to the increase of the sample capacity. The fact that the running time of HACO-B increases slowly suggests that HACO-B is capable of handling very large data sets, and is a more promising algorithm for learning BNs.

4.4. Comparing HACO-B with other algorithms

To evaluate roundly the new algorithm, we compare the solution performance of different algorithms on some Alarm data sets, where Alarm 1000, Alarm 2000, Alarm 5000 are derived from [11], and Alarm 10000 is derived from [3]. Our main objective in this section is to determine whether HACO-B is more efficient

Table 6
Comparisons of the *K2* score and running time for different l_{step}.

l_{step}	Best score	Worst score	Average score	Running time (s)
1	−23781.97	−23782.15	−23782.04 ± 0.02	105.16 ± 5.63
5	−23781.97	−23783.24	−23782.16 ± 0.12	100.02 ± 8.28
10	−23781.97	−23784.25	−23782.59 ± 0.29	101.86 ± 7.14
15	−23781.97	−23784.26	−23782.65 ± 0.29	96.15 ± 4.03
20	−23781.97	−23784.53	−23782.80 ± 0.35	96.08 ± 5.16
25	−23782.06	−23784.44	−23782.99 ± 0.33	88.08 ± 5.43
30	−23782.06	−23784.87	−23782.49 ± 0.27	92.73 ± 4.27
35	−23782.06	−23793.17	−23783.78 ± 1.08	85.49 ± 3.67
40	−23782.06	−23800.09	−23786.22 ± 1.95	90.66 ± 4.26
45	−23782.06	−23785.44	−23782.83 ± 0.38	90.01 ± 4.34

Table 7
The results for two algorithms on some benchmark data with different capacities.

Sample capacity	Statistic	Algorithm	
		ACO-B	HACO-B
Alarm 2000	K2	−9717.64 ± 0.11 (−9717.46)	−9717.46 ± 0.00 (−9717.46)
	A.	3.20 ± 0.13 (3)	3.0 ± 0.0 (3)
	D.	1.0 ± 0.0 (1)	1.0 ± 0.0 (1)
	I.	1.6 ± 0.4 (1)	1.0 ± 0.0 (1)
	It.	59.3 ± 7.55 (30)	42.0 ± 6.28 (20)
	Time (s)	95.71 ± 5.76 (67.06)	40.96 ± 3.55 (26.20)
	Total Num.	79.14E05 ± 0.57 (79.66E05)	71.77E05 ± 1.16 (66.91E05)
	Practical Num.	36108.2 ± 174.66 (36615)	28813.6 ± 327.47 (27363)
Alarm 3000	K2	−14402.01 ± 0.36 (−14401.29)	−14401.29 ± 0.0 (−14401.29)
	A.	2.30 ± 0.33 (2)	2.0 ± 0.0 (2)
	D.	1.0 ± 0.0 (1)	1.0 ± 0.0 (1)
	I.	2.3 ± 0.3 (2)	2.0 ± 0.0 (2)
	It.	72.10 ± 6.32 (61)	40.80 ± 5.43 (20)
	Time (s)	196.93 ± 9.35 (177.66)	54.11 ± 4.65 (36.81)
	Total Num.	82.20E05 ± 0.70 (83.60E05)	66.82E05 ± 0.89 (63.94E05)
	Practical Num.	40200.9 ± 311.24 (40490)	29604.80 ± 205.15 (28646)
Alarm 4000	K2	−19099.64 ± 0.65 (−19098.41)	−19098.70 ± 0.17 (−19098.41)
	A.	2.40 ± 0.22 (2)	2.0 ± 0.15 (2)
	D.	1.0 ± 0.0 (1)	1.1 ± 0.10(1)
	I.	2.5 ± 0.43 (2)	1.50 ± 0.17 (2)
	It.	66.70 ± 6.07 (60)	62.3 ± 7.05 (40)
	Time (s)	247.79 ± 10.97 (256.45)	104.22 ± 16.02 (158.70)
	Total Num.	82.51E05 ± 1.58 (76.27E05)	58.61E05 ± 1.65 (57.74E05)
	Practical Num.	40475.8 ± 512.67 (42122)	25380.5 ± 137.88 (25858)
Insurance 10000	K2	−57591.19 ± 8.13 (−57568.43)	−57580.01 ± 0.32 (−57567.62)
	A.	5.80 ± 1.35 (3)	3.70 ± 0.30 (3)
	D.	9.80 ± 0.48 (8)	8.60 ± 1.63 (8)
	I.	7.90 ± 1.35 (5)	5.60 ± 0.31 (5)
	It.	44.30 ± 7.20 (60)	82.1 ± 4.38 (72)
	Time (s)	296.07 ± 19.90 (337.22)	235.78 ± 17.66 (187.53)
	Total Num.	37.74E05 ± 0.72 (36.47E05)	32.54E05 ± 0.94 (29.13E05)
	Practical Num.	25577.4 ± 412.14 (25587)	18995.6 ± 231.34 (18871)
Asia 1000	K2	−9800.13 ± 0.0 (−9800.13)	−9800.13 ± 0.0 (−9800.13)
	A.	0.0 ± 0.0 (0)	0.0 ± 0.0 (0)
	D.	0.0 ± 0.0 (0)	0.0 ± 0.0 (0)
	I.	1.0 ± 0.0 (1)	1.0 ± 0.0 (1)
	It.	17.90 ± 4.09 (10)	22.0 ± 2.0 (20)
	Time (s)	4.85 ± 0.30 (2.64)	2.78 ± 0.09 (2.48)
	Total Num.	1.22E05 ± 0.02 (1.17E05)	0.69E05 ± 0.01 (0.59E05)
	Practical Num.	457.60 ± 4.32 (451)	325.33 ± 3.75 (304)

and effective than some state-of-the-art approaches on comparable performances. We compare the performance of HACO-B with HEA [11], MDLEP [8], PheGT$_2$^{R} [25], BNPC [3], WinMine and Toolkit [26]. In light of the MDL score, MDLEP [8] applies evolutionary programming for network learning. HEA is an improved algorithm of MDLEP, which adopts both, CI tests to reduce the size of the search space, and a new merge operator, to further enhance search efficiency and effectiveness. PheGT$_2$^{R} applies a genetic algorithm to learn Bayesian networks from data. Therefore, MDLEP, HEA and PheGT$_2$^{R} algorithms are stochastic search approaches, which need to be executed time after time for each testing instance. BNPC [9] is an algorithm based on the dependence analysis approach, and WinMine Toolkit is a set of tools for the windows environment that build statistical models from data. Both algorithms are deterministic algorithms. Thus, they are only executed once for each testing instance. Since the reported results for HEA, MDLEP, PheGT$_2$^{R} are obtained from forty trails [11], we also run HACO-B forty times. In Table 9, we summarize the results over 40 runs. The figures are an average of 40 runs for HACO-B, HEA, MDLEP and PheGT$_2$^{R}. Numbers in parentheses are the standard deviations.

Comparing with stochastic algorithms such as HEA and MDLEP, HACO-B uses much fewer generations to obtain the final solution

Table 8
Comparisons of the K2 score on different data sets for both algorithms.

K2	Algorithms	Sample capacity									
		1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Best	ACO-B	**−5023.28** (2)	**−9717.46** (8)	**−14401.29** (6)	**−19098.41** (**4**)	**−23781.97** (**6**)	**−28347.11** (**6**)	**−33022.93** (3)	**−37745.28** (3)	**−42361.50** (3)	**−47076.20** (7)
	HACO-B	**−5023.28** (**4**)	**−9717.46** (**10**)	**−14401.29** (**10**)	**−19098.41** (**4**)	**−23781.97** (3)	**−28347.11** (1)	**−33022.93** (**7**)	**−37745.28** (**4**)	**−42361.50** (**3**)	**−47076.20** (**8**)
μ	ACO-B	−5024.14	−9717.64	−14402.01	−19099.64	**−23782.17**	**−28347.16**	−33023.92	−37745.93	**−42361.81**	−47076.31
	HACO-B	**−5023.79**	**−9717.46**	**−14401.29**	**−19098.70**	−23782.38	−28347.97	**−33023.48**	**−37745.72**	−42361.92	**−47076.23**
σ	ACO-B	0.34	0.11	0.37	0.65	**0.13**	**0.03**	0.49	0.49	**0.17**	0.11
	HACO-B	**0.23**	**0.0**	**0.0**	**0.17**	0.23	0.26	**0.26**	**0.22**	**0.18**	**0.02**

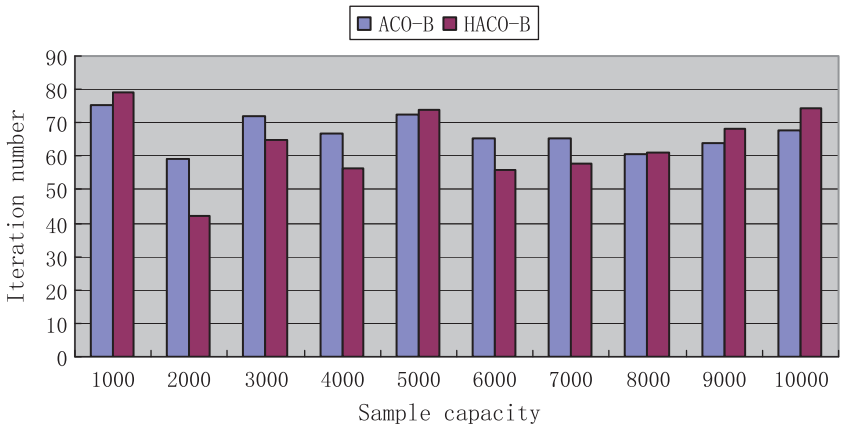

Fig. 6. Comparison of the iteration numbers on Alarm for both algorithms.

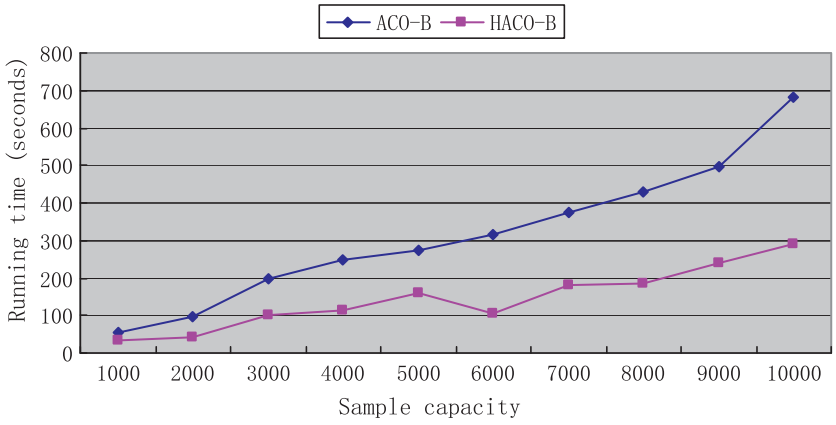

Fig. 7. Comparison of the time performance on Alarm for both algorithms.

Table 9
Performance comparison among HACO-B, HEA, MDLEP, PheGT$_2^R$, BNPC, and WinMine on Alarm data sets.

Performance	Algorithms	Data sets			
		Alarm 1000	Alarm 2000	Alarm 5000	Alarm 10000
The average generation	HACO-B	**62.4 (3.06)**	**57.45 (3.12)**	**50.4 (3.0)**	61.29 (2.95)
	HEA	913.1 (1260.5)	206.8 (155.1)	331.0 (465.2)	877.3 (817.9)
	MDLEP	4301.2 (654.3)	4046.6 (634.1)	3946.3 (651.2)	4523.8 (482.1)
	PheGT$_2^R$	62.8 (56.6)	86.8 (111.3)	61.7 (68.4)	**55.8 (36.7)**
	BNPC	–	–	–	–
	WinMine	–	–	–	–
The average structure difference	HACO-B	**10.58 (0.48)**	**4.70 (0.42)**	**4.75 (0.18)**	**5.48 (0.73)**
	HEA	10.8 (2.1)	7.6 (0.6)	6.1 (0.4)	8.9 (5.0)
	MDLEP	19.4 (4.2)	12.9 (4.9)	10.7 (4.9)	17.5 (6.9)
	PheGT$_2^R$	17.0 (3.1)	12.2 (3.1)	10.7 (3.8)	10.2 (5.2)
	BNPC	17	13	8	9
	WinMine	34	30	23	25

for all data sets. It is also competitive with PheGT$_2^R$ on the generations. Moreover, HACO-B can always find better network structures for all the data sets in terms of the structure difference. The standard deviations are much less for all the data sets in terms of both average generation and structure difference. When we compare HACO-B with BNPC and WinMine, the average structure differences of HACO-B are smaller than those of the two deterministic algorithms for all the data sets. Thus, in light of the average generation and the structure difference, HACO-B is the most outstanding algorithm which can learn the Bayesian networks without a node ordering.

After comparing the performance of different algorithms and focusing on HACO-B versus ACO-B, it is evident that HACO-B is more efficient and effective than ACO-B. Most importantly, HACO-B runs significantly faster than ACO-B. Moreover, HACO-B performs better than HEA, MDLEP, PheGT$_2^R$, BNPC, and WinMine on the solution quality for all data sets.

5. Conclusion

In this paper, we propose a new algorithm, HACO-B, for learning Bayesian networks effectively and efficiently. The algorithm combines dependency analysis, ant colony optimization (ACO) and the simulated annealing strategy. Based on the mechanism of learning BNs by ACO, three new improvements are described. Empirical results illustrate that the new algorithm is superior in terms of the

computational time on all data sets we have tested, while it can also achieve better solution quality on most data sets. Most importantly, our algorithm greatly enhances the convergence speed on large scale data sets compared to the original ACO-B algorithm. We have also presented the performance comparison between HACO-B, HEA, MDLEP, PheGT$_2^R$, BNPC, and WinMine, and found that HACO-B can discover better BNs effectively.

The three new strategies used with HACO-B yield improvements in the reduction of search spaces, the enhancement of the exploration ability of search algorithms and the adjustment of local optimizations, all of which are equally significant for stochastic search algorithms to solve other optimization problems. Our future work is to extend our study to more complex problems in learning BNs, e.g., problems with incomplete data, hidden variables and multi-relational data.

Acknowledgments

This work is partly supported by the NSFC major research program (No. 60496322, 60496327), NSFC (No. 60825203), and the Beijing Natural Science Foundation (No. 4102010).

References

[1] D. Heckerman, A tutorial on learning Bayesian networks, in: M.I. Jordan (Ed.), Learning in Graphical Models, 1996, pp. 301–354.
[2] M. Luis, D. Campos, J. Huete, A new approach for learning belief networks using independence criteria, International Journal of Approximate Reasoning 24 (1) (2000) 11–37.
[3] J. Cheng, R. Greiner, J. Kelly, D. Bell, W. Liu, Learning belief networks from data: an information theory based approach, Artificial Intelligence 137 (2002) 43–90.
[4] J. Suzuki, Learning Bayesian belief networks based on the minimum description length principle: basic properties, IEICE Transactions on Fundamentals E82 (10) (1999) 2237–2245.
[5] D.Y. Liu, F. Wang, Y.N. Lu, W.X. Xue, S.X. Wang, Research on learning Bayesian network structure based on genetic algorithms, Journal of Computer Research and Development 38 (8) (2001) 916–922.
[6] M.C. Luis, M.F. Juan, A.G. Jose, M.P. Jose, Ant colony optimization for learning Bayesian networks, International Journal of Approximate Reasoning 31 (2002) 291–311.
[7] P. Larranaga, M. Poza, Y. Yurramendi, R.H. Murga, M.H. Kuijpers, Structure learning of Bayesian networks by genetic algorithms: a performance analysis of control parameters, IEEE Transactions on Pattern Analysis and Machine Intelligence 18 (9) (1996) 912–925.
[8] M.L. Wong, K.S. Leung, Using evolutionary programming and minimum description length principle for data mining of Bayesian networks, IEEE Transactions on Pattern Analysis and Machine Intelligence 21 (2) (1999) 174–178.
[9] A.G. Jose, M.P. Jose, Searching the best elimination sequence in Bayesian networks by using ant colony optimization, Pattern Recognition Letters 23 (1–3) (2002) 261–277.
[10] J.Z. Ji, H.X. Zhang, R.B. Hu, C.N. Liu, A Bayesian network learning algorithm based on independence test and ant colony optimization, Acta Automatica Sinica 35 (3) (2009) 281–288.
[11] M.L. Wong, K.S. Leung, An efficient data mining method for learning Bayesian networks using an evolutionary algorithm-based hybrid approach, IEEE Transactions on Evolutionary Computation 8 (4) (2004) 378–404.
[12] I. Tsamardinos, L.E. Brown, C.F. Alieris, The max–min hill-climbing Bayesian network structure learning algorithm, Machine Learning 65 (1) (2006) 31–78.
[13] J.R. Alcob, Incremental hill-climbing search applied to Bayesian network structure learning, in: Proceedings of the 15th European Conference on Machine Learning, IEEE, Pisa, Italy, 2004.
[14] M. Dorigo, V.A. Maniezzo, Colorni, The ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics, Part B 26 (1) (1996) 29–41.
[15] M. Dorigo, L.M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Transactions Evolutionary Computation 1 (1) (1997) 53–66.
[16] C. Blum, M. Dorigo, Search bias in ant colony optimization: on the role of competition-balanced systems, IEEE Transactions Evolutionary Computation 9 (2) (2005) 159–174.
[17] C. Blum, M. Dorigo, The hyper-cube framework for ant colony optimization, IEEE Transactions on Systems, Man, and Cybernetics 34 (2) (2004) 1161–1172.
[18] T. Stutzle, M. Dorigo, A short convergence proof for a class of ant colony optimization algorithms, IEEE Transactions Evolutionary Computation 6 (4) (2002) 358–365.
[19] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization: artificial ants as a computational intelligence technique, IEEE Computational Intelligence Magazine 11 (2006) 28–39.
[20] M. Dorigo, T. Stutzle, Ant Colony Optimization, The MIT Press, 2004.
[21] D. Angus, C. Woodward, Multiple objective ant colony optimization, Swarm Intelligence 3 (1) (2009) 69–85, 22 (special issue on ant colony optimization).
[22] Z.J. Lee, C.Y. Lee, S.F. Su, An immunity-based ant colony optimization algorithm for solving weapon-target assignment problem, Applied Soft Computing 2 (2002) 39–47.
[23] Z.J. Lee, S.F. Su, C.C. Chuang, K.H. Liu, Genetic algorithm with ant colony optimization (GA-ACO) for multiple sequence alignment, Applied Soft Computing 8 (2008) 55–78.
[24] G.F. Cooper, E. Herskovits, A Bayesian method for the induction of probabilistic networks from data, Machine Learning 9 (1992) 309–347.
[25] C. Cotta, J. Muruzbal, Toward a more efficient evolutionary induction of Bayesian networks, in: Proceedings of the 7th International Conference on Parallel Problem Solving From Nature, Granada, Spain, 2002, pp. 730–739.
[26] D.M. Chickering, The WinMine Toolkit, Microsoft Research, Redmond, WA, 2002 (MSR-TR-2002-103).