Proceedings of the 2007 IEEE Symposium on
Artificial Life (CI-ALife 2007)

Research on Structure Learning of Dynamic Bayesian Networks by Particle Swarm Optimization

Heng Xing-Chen, Qin Zheng, Tian Lei, Shao Li-Ping

School of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China.

Abstract-**A new approach to learning structure of dynamic Bayesian networks (DBNs) is proposed in this paper. This approach is based on particle swarm optimization (PSO) algorithm. We start by giving a fitness function based on expectation to evaluate possible structure of DBNs by converting incomplete data to complete data using current best DBN of evolutionary process. Next, the definition and encoding of the basic mathematical elements of PSO are given and the basic operations of PSO are designed which provides guarantee of convergence. Next, samples for the incomplete training set and test set are generated from a known original dynamic Bayesian network with probabilistic logic sampling. Next, the structure of DBN is learned from incomplete training set using improved PSO algorithm steps. Finally, the simulation experimental results also demonstrate this new approach's efficiency and good performance in terms of predictive accuracy for test set.**

I. INTRODUCTION

A Bayesian Network (BN) is a convenient graphical way to describe statistical dependencies between a set of variables. It combines graph theory with probability to express complex uncertainty among random variables. Dynamic Bayesian Networks (DBNs) is a species of Bayesian networks (BNs) designed to model stochastic temporal processes, which models the stochastic evolution of a set of random variables over time [1]. Owing to DBNs' significant advantages in describing nonlinear, temporal, evolving and uncertain relationships and strong ability of probabilistic inference, studies on modeling, learning and inference of DBNs have been developed widely [4]. And also, DBNs have been used for many purposes in deferent fields [2, 5, 7]. The most common approach to building dynamic Bayesian networks is to elicit knowledge from an expert. This works well for smaller networks, but when the number of variables becomes large, elicitation can become a tedious and time-consuming affair. There may also be situations where the expert is either unwilling or unavailable. Whether or not experts are available, if there are data building the model from data is a feasible modeling method.

When data are complete, learning the structure of a dynamic Bayesian network can be decomposed into the problem of learning two very simple networks, prior network and transition network because the fitness function used to evaluate structures exists as a closed form expression. However, the data are usually incomplete in most real life applications since we do not have complete observability of the process we want to model. One major complicated factor for incomplete data is that the fitness function exists as a closed form expression for complete data but not for incomplete data. Hence the problem of learning the structure of DBN from incomplete data is much more difficult than for learning from complete data.

In 1998 Friedman expends SEM (Structural EM) to the DBN structure learning in the presence of missing data and hidden variables. But it has been noted that the search space of structure is large and multimodal, and EM as a deterministic search algorithm is prone to find local optima. An obvious choice to combat the problem of "getting stuck" on local maxima is to use a stochastic search method. This paper explores the use of particle swarm optimization (PSO) algorithms for learning dynamic Bayesian networks from incomplete data. Our choice was partly motivated by the work of Clerc et al. [10]. Network structures are especially amenable for evolutionary algorithms since the substructures of the network behave as building blocks so we can evolve higher fit structures by exchanging substructures of parents with higher fitness.

We'll begin by briefly describing dynamic Bayesian networks and the learning problem. Next we will discuss the scoring metric, fitness function, and the landscape. In section 3 we'll make the point for using particle swarm optimization algorithms. Section 4 will describe the design choices we made, to obtain results from a simulation experiment using a known DBN and some results of an empirical study. We'll close in section 5 with a summary of our approach and experiments and discuss our future plans.

II. PROBLEM FORMULATION

We use capital letters, such as *X, Y, Z* for variable names and lowercase letters *x, y, z* to denote specific values taken by those variables. Sets of variables are denoted by boldface capital letters X, Y, Z etc, with sets of values denoted by boldface lowercase letters x, y, z etc. Assume that changes occur between discrete time slices that are indexed by the non-negative integers and that $X = \{X_1, X_2, \ldots, X_n\}$ is a set of attributes that evolves with the process changes. $X_i[t]$ is a random variable that denotes the value of the attribute X_i at time t, and X[t] is the set of random

variables $X_i[t]$. The probability distribution over all the random variables can be represented as follows:

$$P(X[1],\ldots,X[t]) = P(X[1])P(X[2]|X[1])\ldots..P(X[t] \mid X[1],\ldots,X[t-1]) \quad (1)$$

Obviously, such a distribution can be extremely complex. For simplicity of DBN's modeling, learning and inference, two assumptions are introduced. One is Markov assumption I(X[t + 1], {X[1], . . . ,X[t -1]}|X[t]), which means that the state variables at each time slice are only dependent on the state of the last time slice. Given the Markov assumption, the Eq. (1) can be rewritten as:

$$P(X[1],\ldots,X[t]) = P(X[1])P(X[2]|X[1]) \ldots..P(X[t] \mid X[t-1]) \quad (2)$$

Within a finite interval 0, . . . , T , a DBN can be notionally "unrolled" into a BN over X[0], . . . ,X[T].The joint distribution over X[0], . . . ,X[t] is:

$$P_B(X[0],....,X[t]) = P(X[0])\prod_{t=0}^{T-1} P(X[t+1] \mid X[t]) \quad (3)$$

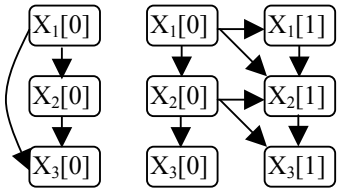
The other assumption introduced is time-invariant assumption, i.e., the transition probability P (**X**[t + 1]|**X**[t]) and the dependence relationship among variables are independent of t and does not vary with time. Given the assumption, a DBN can be simplified further into two parts: A prior network B_0 that specifies a distribution over initial states **X**[0]; and A transition network $B_{\rightarrow}$ over the variables **X**[0] ∪ **X**[1] that is taken to specify the transition probability for all t. Thus, a DBN can be defined by a pair (B_0,$B_{\rightarrow}$), as shown in figure 1(b). The transition probability in figure 1(b) can be computed as follows:

$$P_{B_{-\gg}}(x[1] \mid x[0]) = \prod_{i=1}^{n} P_{B_{->}}(x_i[1] \mid pa(X_I[1])) \quad (4)$$

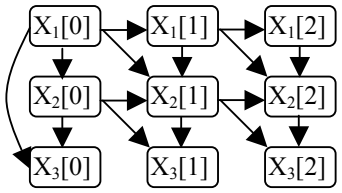
Where **pa** (X_i [1]) denotes the value of the parent node set of X_i[1]. Hence the joint distribution of a DBN over **X** [0], . . . ,**X** [T] can be simply represented as follows:

$$P_B(\text{X [0],}\ldots\text{, X[t] }) = P_{B_0}(\text{X[0]})\, P^{t}_{\mathbf{B}_{-\gg}}(\text{X[1]|X[0]}) \quad (5)$$

The introduction of the above two assumptions makes the DBN' modeling and learning very easy, because only two very simple networks, prior network and transition network, have to be handled.



(a) The prior network and transition network



(b) The DBN extended over 3 time slices.

Fig.1. An example for simplification of DBN

Generally, it is difficult for experts to give the structure of DBN directly, and learning it from data is a feasible modeling method. As a DBN can be defined as two simple networks, B_0 and $B_{\rightarrow}$, learning the structure of a DBN means learning the structure of B_0 and $B_{\rightarrow}$ respectively from data. In addition, the structure learning algorithm can itself be decomposed into searching for structures by using search algorithm and evaluating structures by using scoring metric (fitness function), which aims at finding the best combination of B_0 and $B_{\rightarrow}$ with highest accuracy of generating training set. So the problem of the structure learning can be formally defined as follows:

Input: A training set D of instances of **X**, which contains N observation sequences. The length of the k^{th} sequence is l_k and each case $x_k[0]$, $x_k[1]$,…, $x_k[l_k]$ is given.

Output: A DBN that best matches D. The notion "best matches" is defined using a scoring function.

III. PS_DBN ALGORITHM

Particle swarm optimization (PSO) is an evolutionary computation technique developed by Dr. Eberhart and Dr. Kennedy in 1995, which is inspired by social behavior of bird flocking and fish schooling [2,3]. It has been found to be extremely effective in solving the continuous optimization problem, but now it has been expanded to discrete domain. Though its strict convergence has not been proved, it can be easily modified for any discrete/combinatorial problem for which we have no good specialized algorithm. Therefore, as a combinatorial optimization problem, it is possible to learn the structure of DBN by using PSO algorithm.

PS_DBN algorithm can be expressed simply by the following equation, PS_DBN= (F, X, V, S_{xx} , P_{vv} , M_v , P_{xv} ,, λ , G_{init} , υ), where F is a fitness function, X a space of positions of particles, V velocity set of particles, S_{xx} a substraction operation (position, position), P_{vv} a move operation (position plus velocity), M_v a multiplication operation (coefficient times velocity), P_{xv} a addition operation (velocity plus velocity), λ the swarm size, G_{init} an initial swarm and υ stopping condition.

A. *Fitness Function F*

As a DBN can be decomposed into two BNs, B_0 and $B_{->}$, the fitness function used to evaluate the structure of DBN can be decomposed to evaluate the two BNs respectively.

When data are complete the BIC score and BDe score as two fitness functions for BNs exist in closed form. So we may utilize the score decomposition properties, which facilitate the computation of the BIC score in several ways. First, note that the likelihood is expressed as a sum of terms, where each term depends only on the conditional probability of a variable given a particular assignment to its parents. Thus, if we want to find the maximum likelihood parameters, we can maximize within each family independently. Second, the decomposition implies that we can learn B_0 independently of $B_{->}$. Finally, we can learn $B_{->}$ in exactly the same manner as learning a BN for a set of samples of transitions.

When data are incomplete we no longer have the score decomposition properties. The BIC score and BDe score don't exist in closed form. This is because the formula involves the sufficient statistics, which are not known when data are incomplete. This means that the optimal parameter choice in one part of the network depends on the parameter choices in other parts of the network.

For learning from incomplete data, it is necessary to decrease the computational complexity of the fitness function. So we try to utilize the score decomposition properties by transforming the incomplete data into the complete data. The most commonly used method to alleviate this problem is the *Expectation-Maximization* (EM) algorithm. The E-step of EM uses the currently estimated parameters to *complete* the data by computing the *expected* counts. The M-step then re-estimates the maximum likelihood parameter values as if the expected counts were true observed counts. The central theorem underlying EM's behavior is that each EM cycle is guaranteed to improve the likelihood of the data given the model until it reaches a local maximum.

EM has been traditionally viewed as a method for adjusting the parameters of a fixed model structure. However, the underlying theorem can be generalized to apply to structural as well as parametric modifications. Friedman's *Structural EM* (SEM) algorithm [9] has the same E-step as EM, completing the data by computing expected counts based on the current structure and parameters. In addition to re-estimating parameters, the M-step of SEM can use the expected counts according to the current structure to evaluate any other candidate structure—essentially performing a complete-data structural search in the inner loop. Friedman shows that for a large family of scorings rules, including the BIC score and BDe score, the resulting network must have a higher score than the original. This is true even though the expected counts used in evaluating the new structure are computed using the old structure.

In this paper we will take the BIC score extended for DBNs with EM algorithm.

B. *Encoding PSO Elements for DBN*

The structure of B_0 or $B_{->}$ can be represented as an adjacency list, see Figure 2, where each row represents a variable $\mathbf{X}_i$ and the members of each row, with the exception of the first member, are the parents of $\mathbf{X}_i$, pa($\mathbf{X}_i$). The first member of each row, i.e. the first column of the adjacency list, is the variable $\mathbf{X}_i$.

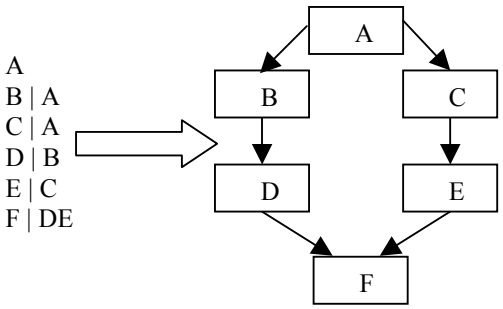


Fig.2. Encoding the structure of a BN

● ***Position of Particles and State space***

As PS_DBN algorithm is designed to find the best structure of BN (B_0 or $B_{->}$) by using PSO, the structure of BN should be encoded into a position of particle.

Although we show the variable $\mathbf{X}_i$ in the figure 2 for clarity, the internal representation encodes its parents only, with the variable being encoded by sequence. The adjacency list can be thought of as a "position" where each pa ($\mathbf{X}_i$) represents a "local position". For example, the "local position" of the variable F can be encoded as [D, E]. Because the logarithm of the scoring metric is the summation of scores for each variable, each local position can be scored separately and added to generate the fitness score for the entire structure. As a DBN composes of two parts, B_0 and $B_{->}$, the corresponding position of particle P should be expressed as $P = (P_0, P_{->})$.

Based on above, the structure of a DBN shown in figure 1 can be encoded as the position of a particle in figure 3:

P_0		$P_{\rightarrow}$	
$X_1[0]$ |		$X_1[t]$ |	$X_1[t-1]$
$X_2[0]$ |	$X_1[0]$	$X_2[t]$ |	$X_1[t]$ $X_1[t-1]$ $X_2[t-1]$
$X_3[0]$ |	$X_2[0]$ $X_1[0]$	$X_3[t]$ |	$X_2[t]$ $X_2[t-1]$

Fig.3. Encoding the structure of a DBN

So the search space is enormous. A local position can range from no parents to n-1 parents, where n is the number of variables in the dataset. Thus a local position can take on

$\sum_{i=1}^{k}\binom{n-1}{i}$ possible values where k is the maximum set of parents a variable can have and n is the number of variables in the dataset. So, the search space can be defined as follow:

$$\sum_{j=1}^{n}\sum_{i=1}^{k}\binom{n-1}{i}$$

● ***Velocity***

Definition 1 ***switch operator***

If a specified DBN has n variables, the position of a particle can be expressed as an adjacency list P = ((x_i)), i=1,…,n. We define a switch operator *SO* which, when applied to a position during one time step, gives another position. So, here, *SO* has three types: $+x_i$, $-x_i$ and ϕ. The $+x_i$ denotes adding a variable x_i into original position, $-x_i$ reducing a variable x_i and ϕ null.

Definition 2 ***switch unit***

A sequence composed of one or several switch operators is a switch unit. We denote it by SU.

$$SU = (SO_1, SO_2, ., SO_k) \quad (6)$$

where $SO_1, SO_2, ., SO_k$ are k switch operators and the different orderings of them have different signification. The length of the SU is defined by ||SU|| =k. Each SU is applied to the change of a local position.

Definition 3 ***switch list (velocity)***

A sequence composed of one or several switch units is a switch list. We denote it by SL. Here, actually, the number of switch units of each switch list is equal to the number of variables *n* of BN. The length of the SL is defined by $||SL|| = \sum_{i=1}^{n} || SU_i ||$. A velocity *V* is then defined by

$$V = SL = (SU_1, SU_2, ., SU_n) \quad (7)$$

where $SU_1, SU_2, ., SU_n$ are n switch units and each SL or *V* is applied to the change of a global position.

Definition 4 *equivalent set of switch list*

If different switch lists are equivalent (same result when applied to any position), the set of them is called equivalent set of switch list.

C. *Designing Operations for PSO*

● ***Opposite of a velocity***

It means to do the same switch as in original *SL*, but with reverse operator. For example, - ((-A), (+B-C)) = ((A), (-B+C)). It is easy to verify that we have - (- *SL*) =*SL* (and $SL \oplus -SL \cong \phi$, see below Addition "velocity plus velocity").

● ***Addition (P_{xv}) "position plus velocity"***

Let *P* be a position and *V* a velocity. The position $P' = P + V$ is found by applying the first switch of *V* to *P*, then the second one to the result etc.

Example

$$\begin{cases} P = (\phi, A, A, B, C, D+E) \\ V = ((+B), (-A), (-A+B), (-B+C), (\phi), (-D-E+A)) \end{cases} \quad (8)$$

Applying *V to P,* we obtain successively

$$P' = (B, \phi, B, C, C, A) \quad (9)$$

● ***Substraction (S_{xx}) "position minus position"***

Let P_1 and P_2 be two positions. The difference $P_2 - P_1$ is defined as the velocity *V*, found by a given algorithm, so that applying *V* to P_1 gives P_2. The condition "found by a given algorithm" is necessary, for, as we have seen, two velocities can be equivalent, even when they have the same size. In particular, the algorithm is chosen so that we have $P_1 = P_2 \Rightarrow V = P_2 - P_1 = \phi$

● ***Addition (P_{vv}) "velocity plus velocity"***

Let V_1 and V_2 be two velocities. In order to compute $V_1 \oplus V_2$ we consider the switch list which contains the first switch unit of V_1, followed by the first switch unit of $V2$, then the second switch unit of V_1, followed by the second switch unit of $V2$ etc. For example, ((-A), (-B+C)) $\oplus$ ((-B+A), (-B+D)) = ((-A-B+A), (-B+C-B+D)). In general, we "contract" it to obtain a smaller equivalent velocity. For example, ((-A-B+A), (-B+C-B+D)) = ((-B), (+C+D)). In particular, this operation is defined so that $V \oplus -V = \phi$. So, we can have the following definition:

Definition 5 *basic switch list*

The switch list of equivalent set of switch list, which contains the least switch operators, is defined as basic switch list. Each velocity is a basic switch list.

● ***Multiplication (M_v) "coefficient times velocity"***

Let α be a real coefficient and *V* be a velocity. There are different cases, depending on the value of α.

Case $\alpha = 0$

We have $\alpha V = \phi$

Case $\alpha \in [0,1]$

We just "shrink" *V*. Let $||\alpha V||$ be the greatest integer smaller than or equal to $\alpha ||V||$. So we define $\alpha V = ((SO_1, .., SO_k)_1, ..., (SO_1, .., SO_k)_n), k \uparrow_1^{(||\alpha V||/n)}$

Case $\alpha > 1$

It means we have $\alpha = d + \alpha'$, d is an integer ($d \neq 0$), $\alpha' \in [0,1]$.

So we define $\alpha V = \sum_{i=1}^{d} (\oplus V) \oplus \alpha' V$.

Case $\alpha < 0$

As $\alpha V = (-\alpha) * (-V)$, we only need to consider one of the previous cases.

D. *Control Parameters*

The initial swarm G_{init} , as well as the velocities, can be generated either randomly or by a Sobol sequence generator[6, 7], which ensures that the D-dimensional vectors will be uniformly distributed within the search space.

The swarm size λ should be not kept too big because of the computation time required scoring the fitness function; On the other hand, λ should be not kept too small for improving the diversity of particles of swarm to avoid premature convergence. Hence, we choose λ within [30,100].

The stopping criterion υ for the algorithm is set in term of that when either g_1 generations have been run or when in g_2 successive generations, the value of the fitness function of the best structure corresponds with the average value of the fitness function.

Ⅳ. PS_DBN ALGORITHM FOR DBN

We can now rewrite the formula from the basic PSO algorithm:

$$V_{id}^{k+1} = w * V_{id}^{k} \oplus c_1 * rand() * (P_{id} - X_{id}^{k}) \oplus c_2 * Rand() * (P_{gd} - X_{id}^{k}) \quad (10)$$

$$X_{id}^{k+1} = X_{id}^{k} + V_{id}^{k+1} \quad (11)$$

where i = 1,2,…,N; N is the swarm's size; d represents the d-dimensional search space; w is the inertia weight factor; c_1 and c_2 are two positive constants, called the cognitive and social parameter respectively; $rand()$ and $Rand()$ are two random numbers uniformly distributed within the range [0,1]; V_{id}^{k} is the velocity of particle i at iteration k; X_{id}^{k} is the current position of particle i at iteration k; P_{id} is the best previous position of particle i at iteration k; P_{gd} is the best neighbour's best previous position at iteration k.

The PS_DBN algorithm can be described as follows:

Step 1: Initialize the particle swarm (each particle is given a stochastic initial solution/position and switch list/velocity).

Step 2: If stopping criterion is satisfied, turn to *Step 5*.

Step 3: Calculate the next position X'_{id} (the new solution) according to the current position X_{id} of the particle i.

1) Calculate the difference α by $\alpha = P_{id} - X_{id}$ where α is a basic switch list and is applied to X_{id} to obtain P_{id}.
2) Calculate the difference β by $\beta = P_{gd} - X_{id}$ where β is also a basic switch list.
3) Calculate the velocity V'_{id} in term of the equation (10) and transform V'_{id} into a basic switch list.
4) Calculate the new solution X'_{id} in term of the equation (11).
5) If a better solution is found, update P_{id} .

Step 4: If a better solution is found for the whole swarm, update P_{gd} and turn to *Step 2*.

Step 5: Show the optimal solution.

Ⅴ. EXPERIMENT

For evaluating the behavior of PS_DBN algorithm proposed, we perform the different experimental steps as follows:

Step 1: Begin with a DBN B = (B_0 , $B_{->}$) (structure and conditional probabilities) and simulate it, generating randomly 1000 samples for the training set D and another 1000 for the test set.

Step 2: Using the approach based on PS_DBN algorithm try to obtain the structure of DBN B_S^* from D, which maximize the probability P (D| B_S).

Step3: Evaluate the performance of PS_DBN algorithm by evaluating the accuracy of B_S^* predicting objective probability distribution.

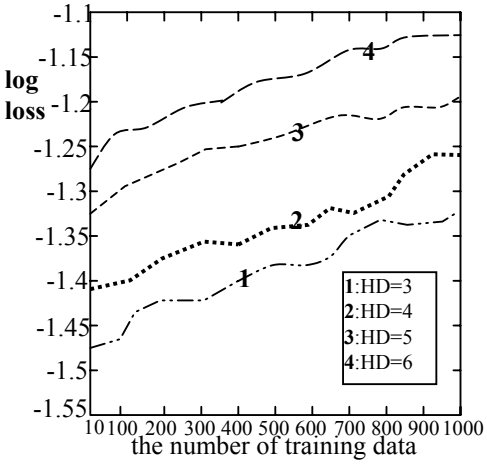
For this experiment we specify initially a DBN known as BAT network that is used to describe the state of automobile running on thruway [8]. It contains 10 state variables, 10 observed variables and several instant variables.

We use probabilistic logic sampling [6], with which we generate 500, 750, 1000 samples each from the original network for training respectively. These training datasets only contain 10 observed variables.

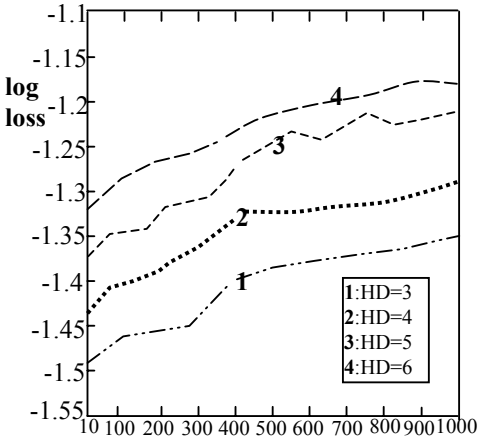
The 3, 4, 5 and 6 hidden variables are introduced respectively into every training dataset in advance to record the change of observed variables and instant variables. The PS_DBN algorithm

is run 10 times respectively for each level of hidden variables, and then the best structure of DBN of 10 times learning is selected as the final learning result of each level of hidden variables.

For the purpose of comparison, all simulations deploy the same parameter settings for the PS_DBN except the swarm size λ and inertia weight w. We try to observe the influence of λ, w and hidden variables on the PS_DBN performance and different w and λ are chosen for simulations. We decide to stop the algorithm when either 1500 generations are reached or when in 200 successive generations, the value of the fitness function of the best structure corresponds with the average value of the fitness function. Finally, we evaluate the performance of result model by using the accuracy of predicting the probability distribution of the objectives through the result model. This concrete process is that we generate 1000 samples as test set containing 10 observed variables from the original network, then calculate the log loss ($\frac{1}{N_{num}} \sum_{i=1}^{N_{num}} \log p(X^i[0],...,X^i[N_i])$) for the test set using the "best" network from each run, which can be seen in figure 4.



(a) $\lambda = 100$, $w \downarrow_{0.9}^{1.2}$



(b) $\lambda = 50$, $w \downarrow_{0.4}^{0.9}$

Fig.4. Comparison of log loss for different number of hidden variables: 3, 4, 5, 6

As can be seen from Fig. 4, the more the hidden variables and the number of training data are introduced, the higher the predictive accuracy becomes.

Ⅵ. CONCLUSION

In this paper we describe a novel approach for learning dynamic Bayesian networks. This problem is extremely difficult for deterministic algorithms and is characterized by a large, multi-dimensional, multi-modal search space. Our approach is based on particle swarm optimization algorithm, which is called PS_DBN algorithm. It is simple and reliable, and it can converge rapidly.

Using simulations of the BAT network, we carry out a performance analysis on the PS_DBN algorithm proposed. The obtained experimental results also prove its efficiency and good performance. Meanwhile, it is confirmed again in this paper that PSO can be applied to solve any combinatorial optimization problem as same as other evolutionary algorithms.

The future important step forwards would be to extend the proposed structure learning approach based on PSO for trying to find out the optimal ordering of these variables in DBN, which is expected to improve the convergence of learning the structure of DBN further. Then, we also plan to adapt the described structure learning approach to hybrid DBNs.

ACKNOWLEDGEMENT

The research reported in this paper is funded by the National "973" Key Basic Research Development Planning Project (2004CB719401).

REFERENCES

[1] Heckerman, D., D. Geiger, et al. (1995). "Learning Bayesian Networks: The Combination of Knowledge and Statistical Data." Machine Learning 20:197-243.

[2] Kennedy, J. Minds and cultures: particle swarm implications. *Socially Intelligent Agents: Papers from the 1997 AAAI Fall Symposium* pp. 67-72. AAAI Press, Menlo Park, CA, 1997.usa, 2001, pp.289-294.

[3] Eberhart, R. C. and Kennedy, J. A new optimizer using particle swarm theory. *Proceeding of the sixth International symposium on micro machine and human science* pp. 39-43. IEEE service center, Piscataway, NJ, Nagoya, Japan, 1995.

[4] J. Pearl, Probabilistic Reasoning in Intelligence Systems: Network of Plausible Inference. San Mateo, Calif.: Morgan Kaufmann , 1998.

[5] R.E. Neapolitan, Probabilistic Reasoning in Expert Systems. Theory and Algorithms. John Wiley & Sons, 1990.

[6] M. Henrion, "Propagating Uncertainly in Bayesian Networks by Probabilities Logic Sampling," *Uncertainty in Artificial Intelligence*, vol. 2, pp. 149-163, 1988.

[7] W.H. Press, W.T. Vetterling, S.A. Teukolsky and B.P. Flannery, Numerical Recipes in Fortran 77,Cambridge University Press: Cambridge, 1992.

[8] Forbes J, T Huang, K Kanazawa, S Russell. The BAT mobile: Towards a Bayesian automated taxi [A]. Proc. of 1995 Intel. Joint Conf. on Artificial Intelligence [C]. Montreal, Canada, 1995.

[9] N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In ICML97, 1997. Vanderbilt University, Morgan Kaufmann Publishers

[10] Clerc, M., Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem, New Optimization Techniques in Engineering, Springer, 2004, 219-239