4-12-2004

An Evolutionary Algorithmic Approach to Learning a Bayesian Network from Complete Data

Ferat Sahin
Rochester Institute of Technology

Jason C. Tillett
Rochester Institute of Technology

Raghuveer Rao
Rochester Institute of Technology

T. M. Rao
SUNY Brockport

Recommended Citation
Ferat Sahin, Jason Tillet, Raghuveer Rao, T. M. Rao, "An evolutionary algorithmic approach to learning a Bayesian network from complete data", Proc. SPIE 5433, Data Mining and Knowledge Discovery: Theory, Tools, and Technology VI, (12 April 2004); doi: 10.1117/12.542371; https://doi.org/10.1117/12.542371

An evolutionary algorithmic approach to learning a Bayesian network from complete data

Ferat Sahin*[a], Jason Tillett[a], Raghuveer Rao[a], T. M. Rao[b]
[a]Rochester Institute of Technology, 79 Lomb Memorial Dr., Rochester NY 14623
[b]SUNY at Brockport, Brockport, NY

ABSTRACT

Discovering relationships between variables is crucial for interpreting data from large databases. Relationships between variables can be modeled using a Bayesian network. The challenge of learning a Bayesian network from a complete dataset grows exponentially with the number of variables in the database and the number of states in each variable. It therefore becomes important to identify promising heuristics for exploring the space of possible networks. This paper utilizes an evolutionary algorithmic approach, Particle Swarm Optimization (PSO) to perform this search. A fundamental problem with a search for a Bayesian network is that of handling cyclic networks, which are not allowed. This paper explores the PSO approach, handling cyclic networks in two different ways. Results of network extraction for the well-studied *ALARM* network are presented for PSO simulations where cycles are broken heuristically at each step of the optimization and where networks with cycles are allowed to exist as candidate solutions, but are assigned a poor fitness. The results of the two approaches are compared and it is found that allowing cyclic networks to exist in the particle swarm of candidate solutions can dramatically reduce the number of objective function evaluations required to converge to a target fitness value.

Keywords: Particle Swarm Optimization, Data Mining, Knowledge Representation, Bayesian Networks, Structural Learning, and Large Databases.

1. INTRODUCTION

This paper explores the use of Particle Swarm Optimization (PSO) technique to solve the NP-Hard[1-4] problem of generating Bayesian Networks (BN) for large datasets. Each variable in the dataset is represented by a node in the BN. The number of possible BN candidates for the dataset is dependent on the number of variables and the number of states in each variable. When the dataset is large and contains a large number of variables, structuring the corresponding BN is computationally expensive. Deciding on a search method for finding the best BN is a key aspect of building BNs from data. Generally, a score–based search algorithm is used for this purpose. For smaller datasets, heuristic search[5] and exhaustive search[6] are used. The heuristic search method requires some knowledge about the order of the variables in the dataset. This knowledge is used to determine the independent variables in the BN. Using the knowledge of the independent variables, arcs are added from the independent variables to dependent variables in the heuristic search. This process results in significant reduction in the number of network candidates explored. An exhaustive search is used to determine the best network by exploring all possible BNs for a dataset. For large datasets, the necessary information about the variables is rarely available for heuristic algorithms to be useful and the number of possible network candidates is too high to perform an exhaustive search. Thus, to construct BNs efficiently, one has to employ specialized optimization algorithms that can handle very large spaces, such as Simulated Annealing[7] (SA), Genetic Algorithms[8] (GAs), and Particle Swarm Optimization[9] (PSO). PSO presents a rather new approach and appears to perform well for large data sets with many local optima. A recent paper[10] reported that PSO outperforms GA and SA in terms of convergence and success in finding global optima.

In the following subsections Bayesian Networks and their structural learning are explored. The PSO technique and its application to structural BN learning are explained in Section 2. Results and analysis are represented in Section 3. Finally, conclusions are summarized in Section 4.

1.1 Bayesian Networks

A Bayesian network consists of the following elements:

- A set of *variables* and a set of *directed edges* between variables,

*feseee@rit.edu; phone 1 585 475 2175; fax 1 585 475 5845

Data Mining and Knowledge Discovery: Theory, Tools, and Technology VI, edited by Belur V. Dasarathy, Proceedings of SPIE Vol. 5433 (SPIE, Bellingham, WA, 2004) · 0277-786X/04/$15 · doi: 10.1117/12.542371

- Each variable contains a finite set of mutually exclusive states,
- The variables coupled with the directed edges construct a *directed acyclic graph* (DAG),
- Each variable A with parents $B_1, B_2, \ldots, B_n$ has a conditional probability table $P(A \mid B_1, B_2, \ldots, B_n)$ associated with it[11].

If the variable A does not have any parents, then the conditional probability table can be replaced by the unconditional probability $P(A)$. A graph is *acyclic* if there is no directed path $A_1 \to A_2 \cdots \to A_n$ such that $A_1 = A_n$. For the directed acyclic graph in Fig. 1, the prior probabilities $P(A)$ and $P(B)$ have to be specified.

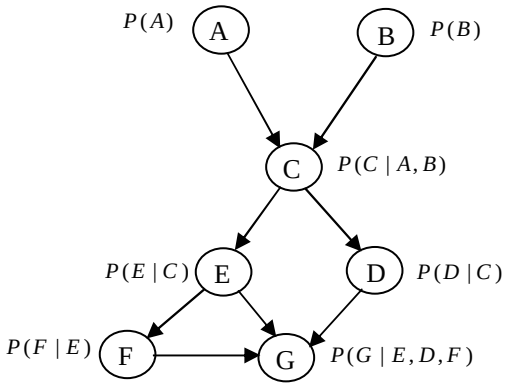

Fig. 1. A *directed acyclic graph* (DAG). The probabilities to specify are shown.

In a BN, let $U = (A_1, A_2, \ldots, A_n)$ be a universe of variables. The chain rule provides a more compact representation of the joint probability $P(U) = P(A_1, A_2, \ldots, A_n)$ to make the probability calculations easier. If the joint probability table $P(U)$ is obtained, then the probabilities $P(A_i)$ as well as the probabilities $P(A_i \mid e)$, where e is evidence, can be calculated. If the number of variables in the network increases, $P(U)$ expands exponentially. Therefore, a more compact representation of $P(U)$, a method of storing information from which $P(U)$ can be computed, is necessary[12]. The following theorem explains this representation.

Theorem 1.1 (The Chain rule.)
For a BN over the universe, $U = (A_1, A_2, \ldots, A_n)$, the joint probability distribution $P(U)$ is the product of all conditional probabilities specified in the BN:

$$P(U) = \prod_i P(A_i \mid pa(A_i)) \tag{2}$$

where $pa(A_i)$ is the parent set of variable A_i [12]. This theorem is very useful in structural Bayesian network learning.

1.2 Structural Learning in Bayesian Networks

This section is devoted to answering the question: how can BNs be learned from data? The process of learning BNs takes different forms depending on whether the structure of the network is known and whether the variables are all observable. The structure of the network can be *known* or *unknown*, and the variables can be *observable* or *hidden* in some or all of the data points. The latter distinction can also be expressed as *complete* and *incomplete* data. Consequently, there are four cases of learning BNs from data; known structure and observable variables, unknown structure and observable variables, known structure and unobservable variables, and unknown structure and unobservable variables. Learning BNs can also be examined as the combination of *parameter learning* and *structure learning*. Parameter learning is the estimation of the conditional probabilities (dependencies) in the network. Structural learning is the estimation of the topology (links) of the network.

Parameter learning, defined as the determination of the conditional probabilities from the available data for a known network structure, is a relatively easier problem and has been studied extensively[1, 13, 14]. On the other hand, the structural BN learning is a much harder problem since the number of candidate networks is intractable when the number

of variables and states in each variable are high. Since we work on large datasets, the size of the data is more important than its completeness. Therefore, we will concentrate on structural learning with observable (complete) data. In this case, the inducer is given the set of variables in the model and asked to select the arcs between them and estimate the parameters. This problem arises in a variety of applications, generally when we are given a new domain with no available domain expert and want to get all of the benefits of a BN model. It also arises in data mining applications, where there are masses of data available and we would like to interpret them. In addition to providing a model that will allow us to predict the behavior of cases that we have not seen, the structure also gives the expert some indication of how attributes are correlated. The algorithms for solving this problem are computationally expensive and basically involve a heuristic search over the space of BN structures.

The problem of reconstructing network topology from fully observable variables has attracted some attention from researchers. This problem, considered a discrete optimization problem, is often solved by greedy search algorithms in the space of network structures[15, 16]. A MAP (Maximum a Posteriori) analysis of the most likely network structure has been studied in the literature[15, 16] when the data are fully observable. The resulting algorithms are capable of recovering fairly large networks from large data sets with a high degree of accuracy[17]. However, they usually adopt a greedy approach for choosing the set of parents for a given node because the problem of finding the best topology is intractable.

There are two main approaches to structure learning in BNs: *Constraint-based* and *score-based*. Constraint–based approach performs tests of conditional independence on the data, and search for a network that is consistent with the observed dependencies and independencies. Score–based defines a score that evaluates how well the (in)dependencies in a structure match the data, and search for a structure that maximizes the score. Constraint-based methods are more intuitive. They follow the definition of a BN more closely. They also separate the notion of the independence from the structure construction. The score-based methods operate on the same principle: a scoring function, that represents how well the network fits the data, is defined for each network structure. The goal is to find the highest-scoring network. An advantage of score-based methods is that they are less sensitive to errors in individual tests. Compromises can be made between the extent to which variables are dependent in the data and the cost of adding the edge[4].

The space of BNs is a combinatorial, consisting of a super-exponential number of structures[5, 18]. The following is a recursively computable expression for the number of possible network structures for a network with n nodes.

$$f(n) = \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{1} 2^{i(n-1)} f(n-i) \tag{3}$$

In general, the problem of finding the highest-scoring network structure is NP-hard[3, 4]. The problem of searching a combinatorial space with the goal of optimizing a function, which involves defining a search space and executing a search algorithm, is very well studied in AI literature. Thus, a structural BN learning algorithm requires the following components be determined:

- Scoring function for candidate network structures.
- The definition of the search space: operators that take one structure and modify it to produce another.
- A search algorithm that does the optimization.

The commonly used scoring functions to learn Bayesian networks are the *Log-likelihood* [4], the *minimal description length* (MDL) score[19] (equivalent to *Bayesian information criterion* [20] (BIC)), and Bayesian score[4, 21]. The log-likelihood function is simply the log of the likelihood function. That is,

$$l(D \mid \mathbf{B}, \theta_{\mathrm{B}}) = \log L(D \mid \mathbf{B}, \theta_{\mathrm{B}}) \tag{4}$$

where D represents the data, $\mathbf{B}$ represents a network candidate, and $\theta_{\mathbf{B}}$ are the parameters of the network $\mathbf{B}$. The log-likelihood is preferred over the likelihood itself because the log turns all the products into sums. Thus, the equation

$$L(D \mid \mathbf{B}, \theta_{\mathrm{B}}) = \prod_{m} P(\mathbf{d}[m] \mid \mathbf{B}, \theta_{\mathrm{B}}) \tag{5}$$

can be re-written as

$$l(D \mid \mathbf{B}, \theta_{\mathrm{B}}) = \sum_{m} \log P(\mathbf{d}[m] \mid \mathbf{B}, \theta_{\mathrm{B}}) \tag{6}$$

where d[m] is a case in the database, $m = 1,2,\ldots,M$.

There are some important facts about the log-likelihood that are worth noting. The log-likelihood increases linearly with the length of data, *M*. The higher scoring networks turn out to be those where the node and the parents are highly correlated. As a result, the network structure that maximizes the likelihood is almost the fully connected network. This is a deficiency of the log-likelihood score. Thus, a score that makes it harder to add edges is necessary. One possible formulation of this idea is called the *MDL score*. It is defined as:

$$Score_{MDL}(\mathbf{B}:D) = l(D \mid \mathbf{B}, \hat{\theta}_{\mathrm{B}}) - \frac{\log M}{2} Dim(\mathbf{B}) - DL(\mathbf{B}) \tag{7}$$

where $Dim(\mathbf{B})$ is the number of independent parameters in $\mathbf{B}$ and $DL(\mathbf{B})$ is the number of bits (the description length) required to represent the structure of $\mathbf{B}$. The MDL score is a compromise between fit-to-data and model complexity. Adding a variable as a parent causes the log-likelihood term to increase, but so does the penalty term. Another commonly used score is called *Bayesian score*. In this case, the network score is evaluated as the probability of the structure, given the data. The Bayesian score has the following form:

$$Score_{BDE}(\mathbf{B}:D) = P(\mathbf{B} \mid D) = \frac{P(D \mid \mathbf{B})P(\mathbf{B})}{P(D)} \tag{8}$$

The probability of that $P(D)$ can be ignored when different structures are compared because it is constant. Therefore, the network structure that maximizes $P(D \mid \mathbf{B})P(\mathbf{B})$, where $\mathbf{B}$ represents a structure. Here, the probability $P(D \mid \mathbf{B})$ can be calculated as

$$P(D \mid \mathbf{B}) = \int P(D \mid \theta_{\mathrm{B}}, \mathbf{B}) P(\theta_{\mathrm{B}} \mid \mathbf{B}) d\theta_{\mathrm{B}} \tag{9}$$

From Equation (9), one can see that the more parameters we have, the more variables we are integrating over. As a result, each dimension causes the value of the integral to go down because the "hill" of the likelihood function is a smaller fraction of the space. Thus, this idea gives preference to networks with fewer parameters. The MDL score can be viewed as an approximation of the Bayesian score. In this paper, we use a scoring function based on Equation (9).

Having discussed several scoring functions, we now return to the problem of finding the network that has the highest score. In other words, a dataset *D*, the scoring function, and a set of possible structures are the inputs to the search algorithm while the desired output is a network that maximizes the score. It can be shown that finding maximal scoring network structures where nodes are restricted to having at most *k* parents is NP-hard[3] for any $k > 1$. Therefore, a heuristic search is generally employed to solve this optimization problem. A search space is defined, where the states in the space are possible structures and the operators denote the adjacency of structures. This space is traversed looking for high-scoring functions to complete the optimization. The obvious operators in the search space are "add an edge", "delete an edge", and "reverse an edge". The search starts with some candidate network, which may be the empty one, or one that some expert has provided as a starting point. Then, the space is searched for a high-scoring network by applying the operators. The most commonly used algorithm is a greedy hill-climbing algorithm, summarized below:

Greedy BN search

Pick a random network structure $\mathbf{B}$ *as starting point*
Pick parameters for each $\mathbf{B}$
Compute score for $\mathbf{B}$
Repeat
  Let $\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_m$ *be the successor networks of* $\mathbf{B}$ *(i.e., operations on* $\mathbf{B}$*)*
  Pick parameters for each $\mathbf{B}_i$
  Compute score for each $\mathbf{B}_i$
  Let $\mathbf{B}'$ *be the highest scoring network among all successor networks*
  If score ($\mathbf{B}'$*) > score (*$\mathbf{B}$*)*
    Then let $\mathbf{B} := \mathbf{B}'$
  Else return ($\mathbf{B}$*)*

Even though the hill-climbing method is commonly used, it has several key problems such as local maxima where all one-edge changes reduce the score and plateaus where a large set of neighboring networks that have the same score.

There are some clever tricks that avoid some of these problems such as TABU-search[22], random restart, and SA[7]. In practice, greedy hill climbing with random start works quite well. Genetic algorithms are another possible solution to this NP-hard problem. Larranaga used a GA-based approach to find the best the BN structure for the well-known *ALARM* network with 37 variables and 46 arcs[23].

The large datasets such as gene data and census data have very high number of possible data cases since each variable has at least two states. Thus, constructing a BN for large datasets becomes very difficult. In this paper, we present a method based on PSO, which can successfully search for the best network for a large dataset.

2. METHODOLOGY: PARTICLE SWARM OPTIMIZATION

The PSO approach utilizes a cooperative swarm of particles, where each particle represents a candidate solution to the problem, to explore the space of possible solutions to the optimization problem of interest. Each particle is randomly (or heuristically) initialized and then allowed to 'fly'. At each step of the optimization, each particle is allowed to evaluate its own fitness and the fitness of its neighboring particles. Each particle can keep track of its own solution, which resulted in the best fitness as well as see the candidate solution for the best performing particle in its neighborhood. At each optimization step, each particle adjusts its candidate solution (flies) according to,

$$\begin{aligned} v(t+1) &= v(t) + \phi_1(x - x_p) + \phi_2(x - x_n) \\ x(t+1) &= x(t) + v(t+1) \end{aligned} \tag{10}$$

Subscripts for particle index and dimensionality have been left off of the Equation (10), which may be interpreted as the 'kinematical' equations of motion for one of the particles. The variables in Equation (10) are summarized in Table 1.

Table 1. List of variables used in the equations

v	The particle velocity.
x	The particle position (test solution).
t	Time.
ϕ_1	A uniform random variable usually distributed over [0, 1].
ϕ_1	A uniform random variable usually distributed over [0, 1].
x_p	The particle's position (previous) that resulted in the best fitness so far.
x_n	The neighborhood position that resulted in the best fitness so far.

Equation (10) can be interpreted as follows. Particles use information about their previous best positions and their neighbor's best positions to maximize the probability that they are moving toward a region of space that will result in a better fitness[9].

Particle swarm optimization can be applied to any problem as long as the fitness function can be defined for the particles. The process is similar to GAs in the sense that the algorithm becomes problem independent after a particle is defined based on the problem. In this paper, each particle will represent a BN. The next section explores the application of PSO to structural BN learning.

2.1 Particle Swarm Optimization for Bayesian Network Discovery

2.1.1 The PSO Ingredients

A simple list of ingredients is required to implement the PSO technique[9]

- A particle construct that fully represents the solution
- A fitness calculator
- A dynamic response of the swarm, see Equation (10)

The particle must represent a possible solution to the problem. In this application, the particle represents a candidate BN. We choose a binary string where each bit represents whether an edge exists between the nodes indexed by the bit. Assuming no node can be its own parent, the binary string will contain $n(n-1)$ bits. Fig. 2 depicts the PSO particle.

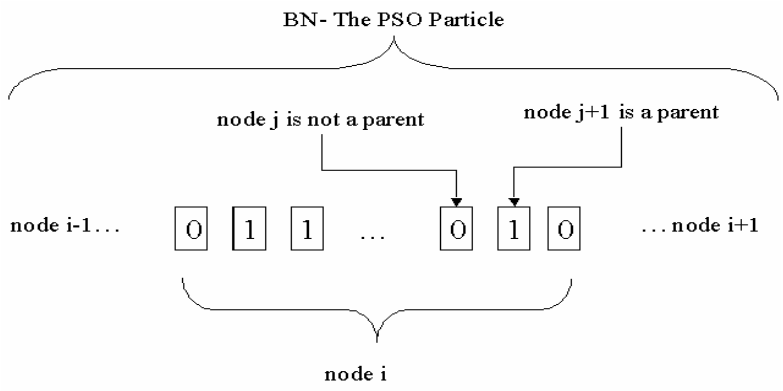

Fig. 2: The PSO Particle for a BN.

To calculate the fitness, we use the scoring of Herskovits[2] given by,

$$Fitness = 1/\log_{10}\left(\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{(r_i-1)!}{(N_{ij}+r_j-1)!}\prod_{k=1}^{r_i}N_{ijk}\right) \tag{11}$$

where r_i is the number of states in node i, the first product is over the nodes in the network, the second one is over the set of permutations of the parents of node i, and the third product is over the states of node i. Also N_{ij} is defined as

$$N_{ij}\sum_{k=1}^{r_i}N_{ijk} \tag{12}$$

In Equation (12) N_{ijk} is an entry in the conditional probability table for node i. The conditional probability table elements contain occurrences of joint instantiations of the parents, (each permutation is indexed with *j*) of node i for which node i is in state k. Therefore, the sum N_{ij} is a total of a column of the conditional probability table, where each column enumerates occurrences of node i in each state for a specific instantiation of the set of parents. Equation (11) can be derived from Equation (9). At each step of the optimization, Equation (10) is used to evaluate the particle velocities, but since our particle is binary[24], the particle's new position will be propagated by evaluating the sigmoid function, σ, of $v(t+1)$ and comparing to a random number, ρ. If $\sigma(v(t+1)) > \rho$ then we set $x(t+1)=1$, otherwise we set $x(t+1)=0$. In summary, our particle is equivalent to a BN. Our fitness is derived from the scoring function given in Equation (11) and our particles move with the dynamics of a binary particle swarm[24].

The PSO approach is, by design, a parallel algorithm. Each particle can evaluate its own fitness independent of the other particles in the swarm. This makes the PSO algorithm ideally suited for execution on a cluster of computers. For large numbers of variables and large datasets, like the *ALARM* network, the fitness calculation is the most computationally expensive aspect of the BN search. We therefore have implemented an MPI (Message Passing Interface) program to perform the BN extraction. We adopt a master-slave topology for our MPI implementation. This topology is adequate specifically because of the synchronous aspect of the PSO algorithm. At the beginning of the program, all contributing processes are initialized by reading the same dataset. The master (process 0) initializes and manages the particle swarm, and distributes particles to the slave processes. Each slave process waits for a particle from the master; upon its receipt, it calculates its fitness and sends it back. When the master has received all fitness results for the swarm, it advances the algorithm by one step and repeats the process of sending out newly evolved particles to the waiting slaves.

As long as the cluster has more processes than there are particles in the PSO swarm, and the complexity of the fitness calculation is approximately constant across all particles in the swarm, the master-slave topology will result in an efficient parallel implementation of the PSO algorithm. This is because, with a high probability, all processes, up to the number of particles in the swarm will be computing fitness values. When there are adequate processes and the complexity of the fitness calculation is constant, all slaves will return particle fitness values at approximately the same

time. This will result in a small idle process time. The maximum idle process time, defined as the length of the time interval between the return of the first and the last fitness calculations, is usually much shorter than the time taken to perform a single fitness calculation. We call this implementation architecture synchronous because all processes must wait for each other to complete their current fitness calculation before the swarm is evolved dynamically.

2.1.2 PSO Algorithm Complexity

If the complexity of the search algorithm is viewed as the number of networks that must be evaluated over the run of the optimization, then the complexity of the PSO algorithm can be written as $O(k \cdot n_p)$ where k is the number of steps in the algorithm and n_p is the number of particles in the swarm. Although the complexity of the PSO search appears to be independent of the number of nodes in the network, the number of steps required to converge to a network will be dependent both on the number of nodes in the network and the number of PSO particles. So a more exact expression of the complexity of the PSO search algorithm for extraction of BNs would be $O(k(n, n_p) \cdot n_p)$ where n is the number of nodes in the network. If we empirically determine a value for k, we can compare the PSO complexity to an exhaustive heuristic algorithm's complexity and thereby get an order of magnitude estimate of the number of nodes, above which it will be more efficient to employ PSO.

A simple exhaustive heuristic[25] is to start with an empty network, and at each step, find the new single arc that when added, results in the best network. An approximate upper bound[26] on the complexity of this search is given by

$$O(\frac{n^4}{2}) \tag{13}.$$

To make the comparison between PSO and this exhaustive heuristic, we constructed a network that is a subset of the nodes of the ALARM network. This subset included 13 nodes and 12 arcs. We constructed this network to compute the exhaustive heuristic in a reasonable time on a PC. Fig. 3 presents the progression of the network fitness as a function of the number of objective function evaluations for both algorithms.

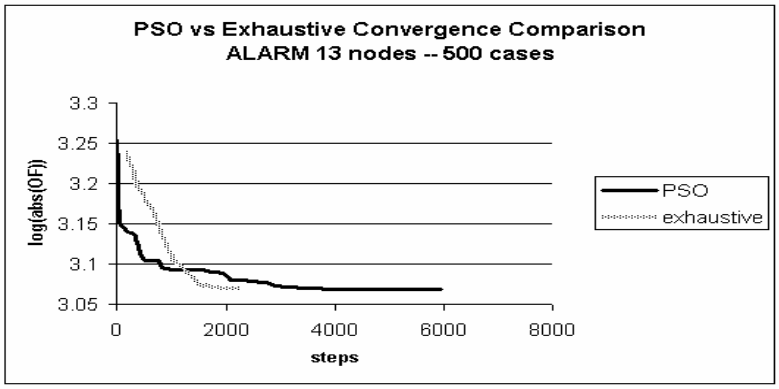

Fig. 3. Complexity comparison, PSO vs. Exhaustive Heuristic

We can conclude from Fig. 3 that both algorithms achieve approximately the same best fitness. PSO is superior to the exhaustive heuristic at achieving good trial networks in fewer steps. The exhaustive heuristic, although capable of performing well for a 13-node network, will not work for extracting the 37-node network, as approximately 10^6 objective function evaluations will be required. Also note that the exhaustive heuristic is prone to stopping on sub-optimal solutions. Since the PSO algorithm was able to achieve good BN extraction for the 37-node ALARM network, it is clear that the complexity of the PSO algorithm, although impossible to enumerate a priori, must be much smaller than the complexity of the exhaustive heuristic.

2.1.3 Cyclic Particles

Our fitness function is designed to handle acyclic graphs only. However, when particles 'fly', they may evolve into cyclic graphs. We identify four possible ways to handle cyclic particles.

- Identify the cyclic particles and render them acyclic by repeated removal of edges.
- Allow the cyclic particles to exist, and assign multiples of a large constant as their fitness depending on how 'bad' they are. For example, we could make the multiple be the number of arcs in their network.
- Allow cyclic particles to exist and assign them a very large constant as their fitness value.
- Use a local optimizer to transform the particle into an acyclic particle.

We have used the first two methods in our simulations. The first method allows the algorithm evaluate more particles by making them acyclic. Thus, the explored search space could be larger. On the other hand, cycles can be allowed in the network and those networks can be assigned an extremely bad fitness value without calculating the network score. Thus, the number of fitness evaluations is reduced drastically even though it can take more PSO steps to converge to the same network structure. Note that we choose to weigh the fitness by the number of arrows in the network to encourage particles to evolve to lower arrow count configurations. We have run some simulations that indicated that if the cycles are allowed the same performance could be achieved with fewer objective function evaluations than the 'break-cycle' strategy.

For our trials where we "break-cycles", the cycles are broken as follows. The network is searched for cycles that result when 2 nodes are each other's parent. One of the nodes in these "2-bit" cycles is randomly selected and its parental relationship to the other node is removed. Most cycles that result after a particle is evolved are 2-bit cycles. After all of the 2-bit cycles are removed, then arcs participating in higher order cycles are randomly removed until the particle is no longer cyclic. After an acyclic particle is found, if the arc count exceeds our maximum, more arcs are removed. We note that after all cycles are removed that the particle will typically have fewer arcs than our allowed maximum.

2.1.4 Particle Initialization

2.1.4.1 Edge Initialization

We perform a heuristic initialization of the particles in the swarm. If there are N nodes in the network, we initialize each particle to contain a randomly selected set of $N/2$ edges. After setting the edges, we test for a cyclic particle. If the particle is cyclic, we scrap it and re-create a new particle. We continue this heuristic until we have successfully initialized all particles in the swarm. We place no limits on the maximum number of parents that a node may have. We do restrict the maximum number of arcs to 74. This restriction has no impact on the particle initialization since we initialize with $N/2$ edges.

2.1.4.2 Velocity Initialization

We initialize each component of each particle's velocity randomly on the interval

$$-v_{\max} \le v_o \le v_{\max} . \tag{14}$$

This initialization will lead to particles having approximately $n(n-1)/2$ arcs after they are moved for the first time. This will ensure that there will be adequate initial exploration of the BN bit string by the particle swarm. We use a maximum velocity of 8, $v_{\max} = 8$. We observed that a lower value of the maximum velocity results is more exploration of the search space. This phenomenon is simply due to the nature of the sigmoid function and the dynamical aspect of the binary particle swarm. We chose the value for *vmax* after executing trial optimizations where we varied *vmax* from 6 to 10. We found that *vmax*=8 performed best.

2.1.4.3 Particle Count, Neighborhood Size and Database Records

We explore the effect of swarm size (particle count) and neighborhood size by leaving these variables as part of our optimization space. The number of records in the database is also varied as described in the next section.

2.1.4.4 Our Grid of PSO Optimization Runs

We ran optimizations varying the number of particles in the swarm, the neighborhood size of the swarm, and the database. We ran each permutation of the above parameters five times to generate statistical data. A neighborhood size of zero implies a global neighborhood where all particles can see fitness values of all other particles in the swarm. A non-zero neighborhood size represents the number of particles on either "side" (in index space), which an individual

particle can probe for fitness values and emulate. The databases used in this research are generated using a Matlab toolbox called BNET[27]. We generated databases of 500, 1000, 2000 and 3000 cases as our test datasets. The optimization was terminated after 1500 steps. This corresponds to 15,000 objective function evaluations for 10 particles and 49,500 objective function evaluations for 33 particles. When a particle moves (evolves) the number of arcs in the network increases. In fact, the increase in the number of arcs is dependent on the progress of the optimization. Early in the optimization, the particles have a high probability of flipping bits in their binary string, as their velocities are still basically random. It is common to have particles with a number of arcs exceeding 600 after they are moved.

3. RESULTS AND ANALYSIS

This section presents the results of the simulations and analysis of the results with comparisons to other work. Table 2 summarizes our results for ALARM network with 37 variables with breaking cycles. Note that these optimizations make no assumptions about the ordering, independence, or leaf node attributes of variables. In Table 2, column 1 reports the number of records in the database used. Column 2 is the number of particles in the swarm. Column 3 is the neighborhood size. We use an index-based neighborhood, where a neighborhood size of two means that each particle can exchange fitness values and positions with two other particles on either side of it in index space, making a total of five particles in the neighborhood. Indexes wrap around so that the highest index particle has particle with index 0 as a neighbor. A neighborhood of size 0 means that all particles can see all other particles; there is one global neighborhood. Each row of the table reports statistics on 5 runs with fixed database, particle count and neighborhood. Column 4 reports the average hamming distance obtained in the five runs. Column 5 reports the best hamming distance. Columns 6 and 7 report the average objective function (AOF) evaluation at the end of the run and the best objective function (BOF) evaluation of the five runs. The objective function (OF) is related to the PSO fitness function by

$$fitness = \frac{1}{OF} \tag{15}$$

where the object function for a network $\mathbf{B}_s$ is given by,

$$OF = \log_{10}(P(D \mid \mathrm{B}_s)) \tag{16}$$

and $P(D \mid \mathbf{B}_s)$ is the parenthetical portion of the denominator of Equation (11).

We find that the 33-particle swarm performs better that the 10-particle swarm. This observation is consistently true across all data sets and neighborhood choices. We attribute this to the fact that the search space is vast. With 37 nodes and no ordering[23], the cardinality of the search space is 3.008^{237}.

Table 2. PSO Applied to the *ALARM* Network with breaking cycles

Cases	P	N	AH	BH	AOF	BOF
500	10	0	67.6	54	-2.507088E+03	-2.474264E+03
500	10	2	68.6	54	-2.580786E+03	-2.518273E+03
500	33	0	48.6	17	-2.414268E+03	-2.340482E+03
500	33	2	46	12	-2.478491E+03	-2.345107E+03
1000	10	0	68.2	50	-4.699734E+03	-4.561657E+03
1000	10	2	76.4	54	-4.952512E+03	-4.813656E+03
1000	33	0	59.6	46	-4.532308E+03	-4.511790E+03
1000	33	2	49.2	32	-4.630997E+03	-4.495897E+03
2000	10	0	73.8	60	-9.104069E+03	-8.993970E+03
2000	10	2	72.6	48	-9.563045E+03	-9.307157E+03
2000	33	0	54.6	26	-8.848883E+03	-8.685584E+03
2000	33	2	54.8	26	-9.138908E+03	-8.858314E+03
3000	10	0	83.2	68	-1.330121E+04	-1.309633E+04
3000	10	2	78.4	68	-1.414775E+04	-1.385497E+04
3000	33	0	73.4	50	-1.306900E+04	-1.294936E+04
3000	33	2	50.2	19	-1.330617E+04	-1.307972E+04

In Table 2, we did not allow cycles in the network by breaking them by removing arrow(s) from the network. As stated before, the cyclic networks can be allowed by assigning bad fitness values to them. Table 3 shows that allowing cycles provides similar solutions with fewer fitness function evaluations. We have run simulations with 33 particles and a global neighborhood (N=0) since these parameters yields the best results for the *ALARM* network.

Table 3. PSO applied to *ALARM* Network with allowing cycles

Cases	P	N	AH	BH	AOF	BOF
500	33	0	76	71	-2.50731E+03	-2.49190E+03
1000	33	0	69	59	-4.55308E+03	-4.46115E+03
3000	33	0	88.6	80	-1.29553E+04	-1.28409E+04
10000	33	0	84	81	-4.19090E+04	-4.16253E+04

As can be seen in Table 3, allowing cycles generate similar results to that of breaking cycles. However, the number of fitness evaluations decreases dramatically since the networks with cycles is assigned bad fitness values without evaluating their fitness. Results in Table 2 and 3 should be compared with the perfect network scores. We have obtained the perfect network score of ALARM network by generating the network in our software and calculating the score of the network. Table 4 presents the fitness functions of the network for 500, 1000, 2000, and 3000 cases. The networks generated by PSO generate sufficiently close fitness values to the target fitness values of the perfect network.

Table 4. Evaluation of the Objective Function for the Original (target) BN

Database cases	Target Objective Function Evaluation
500	-2.34E+03
1000	-4.46E+03
2000	-8.63E+03
3000	-1.26E+04

3.1 PSO Compared to GA for the ALARM Network

Larranaga performed a GA optimization for the *ALARM* network[23]. We chose our database sizes and objective function so as to make the results presented here directly comparable to their results. We identify our PSO results to be directly comparable to their results that are identified as "without order restriction" and "without local optimizer". Order restriction refers to whether parental relationships are contained in the ordering of the nodes in the network. Specifically, nodes with no parents are listed first, followed by nodes with parents already listed. No node is added to the list unless its parents have been added first. The term "local optimizer" refers to what is done to a particle (population member) when cycles exist. If a local optimizer is present, fitness evaluations are performed to determine a way to remove the cycle that results in the best fitness. Tables 15, 16, 17 and 18 contain the results to which we make our comparisons[23].

We compared GA to PSO using the best GA results. For "no order restriction" and "no local optimizer", the best GA results were obtained with an "elitist" population of 50 using a mutation probability of 0.01 and a crossover probability of 0.9. Table 5 reports the average objective function evaluation divided by the target objective function evaluation both for our PSO optimizations and the GA optimizations[23]. We note that PSO extracts a closer approximation to the original network objective function than GA does.

Table 5. Comparison of PSO to GA

Cases	PSO	GA
500	9.6923E-01	9.0570E-01
1000	9.8388E-01	9.2788E-01
2000	9.7519E-01	9.3191E-01
3000	9.6411E-01	9.2113E-01

3.2 Handling cycles in the network

As noted previously, allowing cycles in the PSO does not create inadequate results. However, the PSO algorithm with allowing cycles evaluates extremely low number of objective function evaluations since algorithm does not evaluate the fitness of the network with cycles. Fig. 4 presents the comparison between allowing cycles and breaking cycles for a database of 500 cases, 33 particles and global neighborhood. The simulation results showed that allowing cycles let the PSO converge faster with less number of fitness evaluations. Even though the algorithm reaches the same fitness values with less number of fitness evaluations, it may not converge to better fitness values. We are currently working on combining these methods together so that the algorithm converges fast at the beginning allowing cycles. Then, we can switch to breaking cycles to converge to a better fitness value.

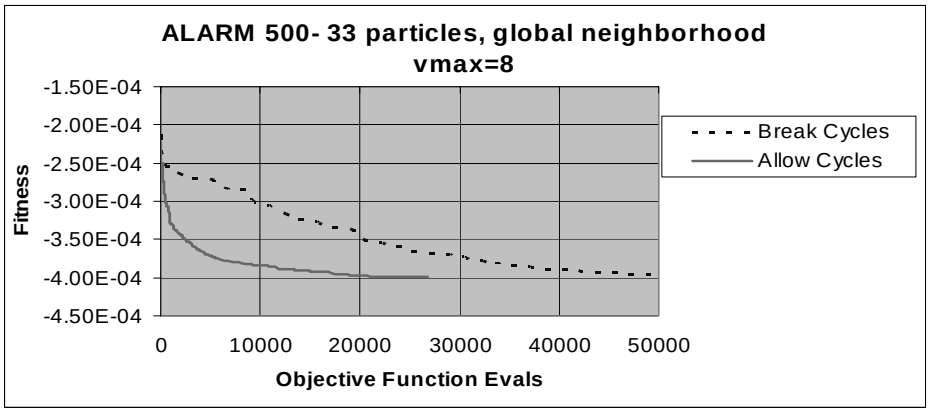

Fig. 4. Allowing cycles versus breaking cycles in PSO.

4. CONCLUSION

In this paper, we present an efficient algorithm for learning the structure of a BN from a large dataset. We have tested the algorithm with well-known *ALARM*[27, 28] dataset with 37 variables and 46 arcs. Results are compared with the *ALARM* network created by GA[23]. PSO algorithm is parallel in nature: all fitness calculations can be performed in parallel. We have exploited this feature and implemented our optimization on a computer cluster consisting of 34 machines.

Parallel computation allows us to perform the network extraction in a fraction of the time, scaling linearly with the number of processes in the cluster, which would be required on a single thread platform. We formulated our trials so that we might directly compare our results to results where a GA is used to extract the BN structure[23]. Tables[23] 15-18 report the results of the genetic algorithm while we report the results of the PSO here in Table 2. Our simulations can be compared against the GA simulations 'without a local optimizer' and 'without order restriction'. We find that in every allowed permutation of the optimization parameters in both techniques, PSO more accurately determines the network structure both in terms of Hamming distance and the objective function fit.

Our investigations into the convergence properties of the PSO algorithm without a local optimizer are encouraging. We were able to extract the original DAG to within a Hamming distance of 12 in the best run. There is sufficient motivation to encourage one to implement a local optimizer for the PSO approach as well as to explore methods for escaping local minima.

ACKNOWLEDGEMENT

This research was conducted with the support of the Gleason Foundation through the Laboratory of Autonomous Cooperative Microsystems (LACOMS) at the Rochester Institute of Technology and the support of the Office of Naval Research in the Machine Intelligence Laboratory at Virginia Tech.

REFERENCES

1. D. J. Spiegelhalter and R. G. Cowell, "Learning in probabilistic expert systems," Bayesian Statistics, vol. **4**, 1992.
2. E. Herskovits, "Computer-based probabilistic network construction," in Medical Informatics: Stanford University, 1991.
3. D. M. Chickering, D. Gieger, and et.al., "Learning Bayesian network is NP-Hard," Micrsoft Research, Redmaond, WA 1994.
4. D. Koller, "Artificial intelligence: Knowledge representation and reasoning under uncertainty, Course material (CS 288)," http://www.stanford.edu/class/cs228/index.html, 1999.
5. G. F. Cooper and E. Herskovits, "A Bayesian method for constructing Bayesian belief networks from databases," *Proceedings of the Conference on Uncertainty in AI*, pp. 86 - 94, 1990.
6. F. Sahin and J. S. Bay, "Structural Bayesian network learning in a biological decision-theoretic intelligent agent and its application to a herding problem in the context of distributed multi-agent systems," *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC 2001)*, pp. 1606 – 1611, 2001.
7. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," Science, vol. **264(5163)**, pp. 1297 - 1301, 1983.
8. J. Holland, *Adaptation in Natural and Artificial Systems*: University of Michigan Press, 1975.
9. J. Kennedy and R. Eberhart, *Swarm Intellignce*: Morgan Kaufmann Publishers, ISBN 1-55860-595-9, 2001.
10. A. Salman, I. Ahmad, and S. Al-Madani, "Particle swarm optimization for task assignment problem," Microprocessors and Microsystems, vol. **26**, pp. 363-371, 2002.
11. F. V. Jensen, *An Introduction to Bayesian Networks*. London, UK: University College London Press, 1996.
12. J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.
13. K. G. Olesen, S. L. Lauritzen, and F. V. Jensen, "aHUGIN: A system for creating adaptive causal probabilistic networks," *Proceedings of the Eighth Conference on Uncertainty in AI (UAI '92)*, Stanford, CA, 1992.
14. D. Spiegelhalter, P. Dawid, S. L. Lauritzen, and R. Cowell, "Bayesian analysis in expert systems," Statistical Science, vol. **8**, pp. 219-282, 1993.
15. G. Cooper and E. Herskovits, "A Bayesian method for induction of probabilistic networks from data," Machine Learning, vol. **9**, pp. 309-347, 1992.
16. D. Heckerman, D. Gieger, and M. Chickering, "Learning Bayesian networks: The combination of knowledge and statistical data," Technical Report MSR-TR-94-09, Microsoft Research, Redmond, WA 1994.
17. S. Russell, J. Binder, and D. Koller, "Adaptive probabilistic networks," Technical Report UCB//CSD-94-824, 1994.
18. R. W. Robinson, "Counting Unlabeled Acyclic Digraphs," in *Lecture Notes in Mathematics,622:Combinatorial Mathematics V.*, C. H. C. Little(Ed.), Ed. New York: Springer-Verlag, 1977.
19. W. Lam and F. Bacchus, "Learning Bayesian belief networks: an approach based on the MDL principle," Computational Intelligence, vol. **10**, pp. 269-293, 1994.
20. G. Schwarz, "Estimation the dimension of a model," Annals of Statistics, vol. **6**, pp. 462-464, 1978.
21. D. Heckerman, "A tutorial on learning Bayesian networks," Technical Report MSR-TR-95-06, Microsoft Research 1995.
22. F. Glover, "Tabu search 1," ORSA Journal on Computing, vol. **1(3)**, pp. 190 - 206, 1989.
23. P. P. Larranaga, M.; Yurramendi, Y.; Murga, R.H.; Kuijpers, C.M.H., "Structure learning of Bayesian networks by genetic algorithms: a performance analysis of control parameters," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. **18**, pp. 912 -926, 1996.
24. J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, pp. 4104-4109, 1997.
25. J. C. Tillett and F. Sahin, "IntelliAgent: Simulation Environment for Cooperating Intelligent Agents," *Proceedings of the IEEE SMC2002 Conference*, vol. 3, 2002.
26. F. Sahin, "A Bayesian approach to the self-organization and learning in intelligent agents," Dissertation: Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg. 2000.
27. K. Murphy, "Bayes Net Toolbox for Matlab", http://www.ai.mit.edu/~murphyk/Software/BNT/bnt.html, 2004.
28. I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper, "The ALARM Monitoring System: A case study with two probabilistic techniques for belief networks," *Proceedings of the Second European Conference on Artificial Intelligence in Medicine*, vol. 38, pp. 247-256, 1989.