Available online at www.sciencedirect.com
ScienceDirect
Parallel Computing 33 (2007) 124–143
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Fault diagnosis for airplane engines using Bayesian networks and distributed particle swarm optimization ☆

Ferat Sahin [a,*], M. Çetin Yavuz [a], Ziya Arnavut [b], Önder Uluyol [c]

[a] *Electrical Engineering, Rochester Institute of Technology, Rochester, NY 14623, United States*
[b] *Computer Science, SUNY Fredonia, Fredonia, NY 14063, United States*
[c] *Aerospace Advanced Technology, Honeywell Inc., Minneapolis, MN 55418, United States*

Available online 9 January 2007

Abstract

This paper presents a fault diagnosis system for airplane engines using Bayesian networks (BN) and distributed particle swarm optimization (PSO). The PSO is inherently parallel, works for large domains and does not trap into local maxima. We implemented the algorithm on a computer cluster with 48 processors using message passing interface (MPI) in Linux. Our implementation has the advantages of being general, robust, and scalable. Unlike existing BN-based fault diagnosis methods, neither expert knowledge nor node ordering is necessary prior to the Bayesian Network discovery. The raw datasets obtained from airplane engines during actual flights are preprocessed using equal frequency binning histogram and used to generate Bayesian networks fault diagnosis for the engines. We studied the performance of the distributed PSO algorithm and generated a BN that can detect faults in the test data successfully.

Keywords: Bayesian networks; Fault diagnosis; Particle swarm optimization; Parallel computing

1. Introduction

Recently, the general problem of fault diagnosis and prognostics has been getting a lot of attention from researchers. Bayesian network (BN)-based fault diagnostics systems have several advantages; they combine graph theory which makes them more intuitive and probability theory which makes them represent uncertainty well. This paper presents a technique, Particle Swarm Optimization (PSO), to construct BN from large datasets as well as how to exploit it for fault diagnosis and predictive maintenance of airplane engines. This approach models the system using the sensor readings rather than characterizing the engine dynamics using the domain knowledge as given in [50]. The large datasets are populated from raw data collected from sensors

☆ This research was conducted with the support of Honeywell Inc. at the Multi-Agents Bio-Robotics Laboratory at Rochester Institute of Technology.
* Corresponding author. Tel.: +1 585 475 2175; fax: +1 585 475 5845.
E-mail addresses: feseee@rit.edu (F. Sahin), yavuzmc@gmail.com (M. Çetin Yavuz), ziya.arnavut@fredonia.edu (Z. Arnavut), onder.uluyol@honeywell.com (Ö. Uluyol).

0167-8191/$ - see front matter
doi:10.1016/j.parco.2006.11.005

of airplane engine during actual flights. After preprocessing, the datasets are first packed then compressed and then fed to our PSO-based BN discovery software running in parallel on a computer cluster of 48 CPUs. The parallelization is achieved by using message passing interface (MPI) in Linux. In this paper, variables, nodes or columns will be used interchangeably. Our datasets, also called as input files, are simple text files, in which the first row contains node names and all the other rows contain integers representing the sensor readings at a given time. Using our PSO-based Bayesian network discovery algorithm, we find a Bayesian network, which can detect faults in airplane engines. After we generate the Bayesian network, we run inference on the BN using a separate set of engine data. Through the inference, our algorithm is able to determine whether the data is coming from a faulty engine. The algorithm calculates a probability of fault for the engine data being tested.

This paper is organized as follows: Section 2 gives background in Bayesian networks, BN learning methods, and literature on fault diagnosis with BN. Section 3 presents the fault diagnosis system with Bayesian network and particles swarm optimization. The experimental results and conclusions are provided in Section 4 and Section 5, respectively.

2. Bayesian networks

Bayesian networks are directed *acyclic* graphs that are constructed by a set of variables coupled with a set of directed edges between the variables. Bayesian networks are very successful in reasoning between the variables via conditional probabilities. A Bayesian network is a graphical model that carries probabilistic information about its nodes. It has a set of variables $U = (A_1, A_2, \ldots, A_n)$ and a network structure represented by a graph of conditional dependencies among the variables in U. A conditional dependency connects a *child* variable with a set of *parent* variables. This dependency is represented by a table of conditional distributions of the child variable given each combination of the values of the parent variables.

In the light of the above information, properties of BNs can be summarized as below:

- It has a set of *variables* and a set of *directed edges* between variables.
- 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 $P(A|B_1, B_2, \ldots, B_n)$ associated with it [23].

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 $A_1 \rightarrow A_2 \cdots \rightarrow A_n$ 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.

In a BN, let $U = (A_1, A_2, \ldots, A_n)$ be a universe of variables. If the joint probability table $P(U) = P(A_1, A_2, \ldots, A_n)$ is obtained, then the probabilities $P(A_i)$ as well as the probabilities $P(A_i|e)$, where e is evidence, can be calculated. If the number of variables in the network increases, $P(U)$ expands exponen-

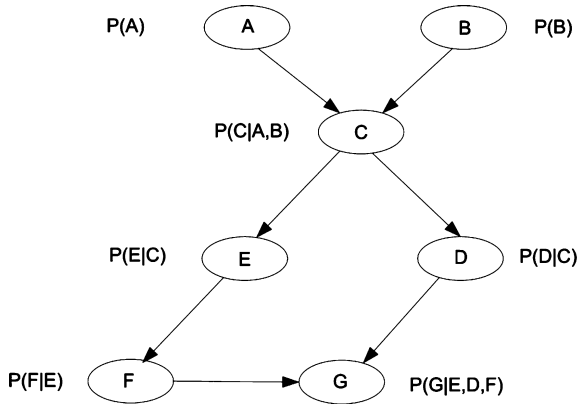

Fig. 1. A *directed acyclic graph* (DAG). The probabilities to specify are shown.

tially. The joint probability table $P(U)$ can be computed from the conditional probabilities defined in a BN if the conditional independencies hold for U using the following theorem [39].

Theorem (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*:

$$P(U) = \prod P(A_i|pa(A_i)) \tag{1}$$

where $pa(A_i)$ *is the parent set of variable* A_i.

Jensen proved this theorem by applying induction on the number of variables in the universe U [39]. For Fig. 1, the joint probability distribution can be calculated as follows:

$$P(A,B,C,D,E,F,G) = P(A)P(B)P(C|A,B)P(E|C)P(D|C)(F|E)P(G|D,E,F) \tag{2}$$

2.1. Learning Bayesian networks

Bayesian network learning can be classified based on whether the structure of the network is known or unknown and the variables (data) can be observable (complete) or hidden (incomplete). Consequently there are four classes 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. The problem we attack fall into the third class in which we try to learn the structure of the BN by using the complete sensor data for fault diagnosis of airplane engines. 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. Structural BN learning is much harder problem compared to parameter learning since the number of candidate networks grows super-exponentially when the number of variables increases [24].

2.1.1. Known structure complete data

In this case, the problem is to calculate the conditional probability tables of each node in the network from the complete data (parameter learning). This is an easier problem and has been studied extensively [38,48,49].

2.1.2. Known structure incomplete data

The problem of learning parameters for a fixed network in the presence of missing values or hidden variables studied by extending and adapting *expectation-maximization* (EM) algorithm [15] and by Gibbs sampling. Both of these algorithms use a basic strategy that is to estimate the missing data on the basis of available data and information about the missing data. Another approach, called *bound and collapse* (BC) [42,47], first bounds the set of possible estimates consistent with the available information by computing the optima of estimates that are gathered from all possible completions of the database constraint by the given pattern of the missing data and then collapses these bounds to a point estimate using information about the assumed pattern of missing data. Genetic algorithm [36] is used to evolve both the missing values and the structures to find an optimal Bayesian network.

2.1.3. Unknown structure incomplete data

This is the most difficult hence the least studied case. Since it is generally not feasible to compute exact solutions to this problem, approximate algorithms generally employed. It is first attacked by a gradient-based algorithm [44] by using *structural* EM (SEM) that optimizes parameters with structure search for model selection. Convergence to a sub-optimal network and need for heavy computation during the learning are major problems with the structural SEM algorithm. Another approach is to use above mentioned BC for incomplete data [47]. The problem of learning the structure of BN from incomplete data is also considered in [15,21,51].

2.1.4. Unknown structure complete data

Our problem falls into this category in which we are given a complete dataset and asked to generate BN that fits the data the best. This problem is in fact a discrete optimization problem in mathematics but can come

in the forms of data mining, system identification, or classification problem in a variety of domains. The algorithms for solving this problem are computationally expensive as the number of structures in the structure space is super-exponential in the number of variables of the system. They mostly involve a heuristic search [12,13,19,21,31]. Genetic algorithm was applied for searching structures by assuming that the ordering between the nodes of the network structures is given. This reduces the structure space significantly and prevents the algorithm from producing illegal BNs (not DAG) [31]. Next the ordering assumption is released and illegal networks are converted to legal ones by randomly eliminating the edges that invalidate DAG conditions. One of the other restricting assumptions in this paper was that a node can have at most four parents (maximum in-degree 4). The "Sparse Candidate" algorithm [16] speeds up learning at cost of a slightly lower final score. This iterative algorithm restricts the parents of each variable belong to a small subset of candidates by using statistical cues from the data. Then a search for a network that satisfies constraints constructed from these cues is done. The learned network is used for selecting better candidates for the next iteration. One of the disadvantages of this algorithm is its poor robustness i.e. and its high sensitivity to prior statistical knowledge from data which could make it learn erroneous networks. Another approach is to reduce the search space by searching among independence-equivalent classes of networks instead [1,9].

Blanco et al. [6] compared two new population-based stochastic search approaches, univariate marginal distribution algorithm (UMDA) and population-based incremental learning (PBIL) in a score + search framework with genetic algorithm (GA) approach. They tested these three algorithms with three different scores: penalized maximum likelihood, marginal likelihood, and information theory-based entropy on three databases of 10,000 cases generated from the *Asia*, the *Alarm*, and the *Water* with 8, 32, and 37 nodes, respectively. They got competitive results from these two new algorithms with respect to GA for both when the ordering is known and not known. When the ordering is taken into account, the learned structures are close to the original network and the score of the network is improved. However, when the ordering is not taken into account the learned structures are different in a large degree with respect to the original structures. Chickering [10] made a significant contribution first by proving the so called "Meek Conjecture" [34] second by providing a two-phase algorithm to find an equivalence class of BNs which is optimal along certain dimensions using a greedy search in the space of equivalence classes, in the limit of an input dataset of infinite size. He starts with an empty model with no arc and in the first phase, the algorithm finds a model that includes the true model by examining all models in the upper boundary of the current model and picking the one with the best score until a local maximum is reached. In the second phase, the algorithm reduces it to the true model by examining all models in the lower boundary of the current model and picking the one with the best score until a local maximum. He successfully applies this algorithm to the synthetic and real-world data but their results were not best in terms of the learning time compared to the other greedy search algorithm.

Peng and Ding [40] extend K2 algorithm [12] so that it does not require a predefined ordering of nodes. They propose a linear parent search method to generate candidate graph and used a new approach to eliminate cycles in the candidate graph. Finally they assess the stability of the network by using structure perturbation and refine it by stability-improvement method. They apply these techniques on the well known alarm dataset [4] and yeast gene expression data set. Friedman and Koller [18] were more interested in computing the Bayesian posterior of a feature, i.e., the total posterior probability of all models that contain it rather then finding a single model. For that purpose they modified *Markov Chain Monte Carlo* (MCMC) method of [32]. They first showed how to efficiently compute a sum over the exponential number of networks that are consistent with a fixed order over network variables. Then they computed both the marginal probability of the data and the posterior of a feature for the given order. After that, they used this result as the basis for their algorithm, which approximates the Bayesian posterior of the feature. Koivisto and Sood [29] proposed an idea of using dynamic programming for exact BN structure discovery. They claimed that their algorithm is the first algorithm with a complexity less than super-exponential with respect to the number of variables. An extension of [17,18] is provided in [29], which computes the Bayesian posterior probability of structural features, such as edges, as well as a single MAP (Maximum a Posteriori) model, by MCMC search over orderings. It is shown in [29] that the equations of [17,18] can be solved as a recursive function over intermediate terms. Their algorithm is applied to synthetic data and not feasible for networks which have more than 25 variables.

Yi-feng and Kim-leng [53] propose a new algorithm called Block Learning Algorithm, by adopting the divide and conquer strategies, for data sets of large number of variables with small number of cases (similar to the datasets used in [16]). The method partitions the variables into several blocks that are overlapped with each other. The blocks are learned individually with some constraints obtained from the learned overlap structures. After that, the whole network is recovered by combining the learned blocks. Their methods are more robust compared to other constraint-based methods.

2.2. Structural Bayesian network learning

There are two main approaches to structure learning in BNs: constraint-based and score-based. In the first one tests of conditional independence on the data are performed, then a search for a network that is consistent with the observed (in)dependencies is done. The score-based methods operate on the same principle: a scoring function, that represents how well it fits the data, is defined for each network structure. The goal is to find the highest-scoring network structure. 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 [20]. In general, the problem of finding the highest-scoring network structure is NP-hard [8]. 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 space of BNs is a combinatorial space, consisting of a super-exponential number of structures [11,43]. The following is a recursively computable expression for the number of possible network structures for a network with n nodes [11].

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

2.2.1. Scoring functions

The three main scoring functions commonly used to learn Bayesian networks are the *log-likelihood* [20], the *minimal description length* (MDL) score [30] (which is equivalent to Schwarz' *Bayesian information criterion* (BIC) [46]), and Bayesian score [20]. The *log-likelihood* function is the log of the likelihood function. That is,

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

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

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

can be re-written as

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

where $d[m]$ is a case in the database, $m = 1, 2, \ldots, M$. The *log-likelihood* increases linearly with the length of data, M. In addition, adding an arc to the network always increases the *log-likelihood*. Thus, the network structure that maximizes the likelihood is almost the fully connected network. In Bayesian networks, the less the number of arcs is the better for the exact inference calculations. 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

$$\text{Score}_{\text{MDL}}(B:D) = l(D|B,\hat{\theta}_B) - \frac{\log M}{2}\text{Dim}(B) - \text{DL}(B) \tag{7}$$

where Dim(B) is the number of independent parameters in B and DL(B) is the number of bits (the description length) required to represent the structure of B. Adding a variable as a parent causes the *log-likelihood* term to increase but so does the penalty term. There will be an edge addition if its increase to the likelihood is worth it.

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:

$$\text{Score}_{\text{BDE}}(B:D) = P(B|D) = \frac{P(D|B)P(B)}{P(D)} \tag{8}$$

The probability of the data, $P(D)$, can be ignored when different structures are compared because it becomes a constant. Therefore, the network structure that maximizes $P(D|B)P(B)$, where B represents a structure, maximizes the score as well. Here, the probability $P(D|B)$ can be calculated as

$$P(D|B) = \int P(D|\theta_{\text{B}}, B)P(\theta_{\text{B}}|B)\text{d}\theta_{\text{B}} \tag{9}$$

From Eq. (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. Therefore, this idea gives preference to networks with fewer parameters. It can be shown that the Bayesian score is a general form of MDL score. In this paper, we use a scoring function based on Eq. (9).

2.2.2. Search algorithms

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 for any $k > 1$. Therefore, a heuristic search is generally employed to solve this optimization problem. First a search space, in which the states represent possible structures and the operators denote the adjacency of structures, is defined. Then, 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 simple greedy hill-climbing algorithm. 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.

Algorithms proposed to find the best network for a complete dataset are presented in Section 2.1.4. As mentioned before, these algorithms assume that ordering of the variables is known and most of them do not scale well with networks with large number of variables (more than 25). In addition, they do not scale well for large datasets such as gene and census data. Our approach is based on particle swarm optimization (PSO) that can successfully search for the best network for a large dataset and for large networks.

2.3. Fault diagnosis with Bayesian networks

Recently, there have been several attempts to use Bayesian networks for fault diagnosis [28,33,35,37,41,45]. When the structure of the BN is known (provided by experts), it is relatively easy to apply it to diagnosis. For example in [28], they proposed a set of nonlinear equations and constraints to be solved for the probabilities to represent a simple BN. It is shown that a BN is an efficient way to combine systemic, expert, and factual knowledge for diagnostics [27]. In 2003, Soroush et al. [35] proposed a BN model for fault detection and

identification in a single sensor as a building block to develop a BN model for all the sensors under consideration. Przytula and Thompson [41] give a procedure for creating BNs from expert knowledge, repair records, and technical manuals for complex systems by using decomposition of the systems into subsystems and later composing it into a whole model for diagnostics. In all of the approaches for fault diagnosis using BNs, it is assumed that the order of variables and/or the BN structure are known. In our application, we are not given any expert input prior to discovering the BN from the engine data. On the contrary, our goal is to provide feedback to engine experts with respect to relationships between the system variables and fault. Initial findings of our approach was promising as stated in [52]. The next section presents the implementation details of our fault diagnosis technique using Bayesian network and particle swarm optimization.

3. Fault diagnosis using Bayesian network and particle swarm optimization

In this section, we present the implementation of the structural Bayesian network learning for fault estimation and diagnosis of airplane engines. Data preprocessing, search algorithm for finding best Bayesian network representing the data the best using particle swarm optimization (PSO), and parallel computing for PSO algorithm using message passing interface in a Linux computer cluster are main components of the implementation.

3.1. Preprocessing

Preprocessing of the raw data is necessary for a variety of reasons such as to extract features from data and to have a finite number of states. The preprocessing has three main steps: adjusting the sampling rates for all the variables, equal frequency data binning (adaptive histograms), re-labeling the data based on the histogram bins.

The sensor data from the airplane engine are stored as structures in numerous MATLAB files. Each file consists of data from sensors measuring the engine or airplane parameters such as temperature, pressure, altitude, flight phase, etc. for a complete flight. These data files are sequentially loaded into MATLAB and the data corresponding to lube system related variables are extracted. Since the sensor readings that are used in generating the training sample are sampled at different sampling periods, we resample the data so that each sensor is resampled with the same rate by either compressing or expanding.

The data corresponding only to the approach phase of the flight data are extracted from the whole flight data. There are two reasons for this choice. The first is to increase the coverage of flight data analysis unlike the takeoff and cruise, the approach phase has not been studied well. The second, and the more critical reason, is that the sensors relevant for lube diagnosis record a broad range of values during the approach phase thereby allowing us to delineate distinct states for the BN nodes. The data from the approach phase are then combined in a large dataset. The data adjustment is then followed by an equal frequency data binning process. This algorithm clusters the data into bins while ensuring that each bin contains fairly equal number of elements. After the equal frequency binning, the original data are represented by the bin numbers associated with them. As a result the maximum number of states in each node has been decreased. Thus, the conditional probability tables in the variables will have manageable sizes because each node (variable) will have small (we chose four) number of states.

3.2. Fault addition

In addition to the raw sensor data collected during the flights, we are provided with the maintenance records of the airplanes showing the faults experienced and regular maintenance. For example if any oil related repair is done on the engine right after a flight, that flight is considered to have a faulty engine data and the file is marked as faulty. Similarly the very first flight after that maintenance is considered to have non-faulty flight data. The raw data files are marked up manually as faulty or not in their file names to automate the fault column population in the table. After the data binning step, a new column (with the name ''Fault'') is added to the data. The values for the fault column entries are binary and determined as follows; if it is coming from a fault raw data file, it is set to 1; otherwise it is set to 0.

3.3. Packing and unpacking data

The maximum number of states in any node is not likely to be greater than 16. Thus, we can pack the data in pairs since 16 different values require four bits to represent each value uniquely. The first row of the file consisting of the name of the nodes is not packed as it includes letters. A specific format is followed and is written to the first row of the packed data file to ease the process of unpacking. This first row includes the number of data elements, the number of nodes, the names of the nodes and an end of line character (EOL) to flag that the packed data elements (integers) are starting after that line. In the process, redundant bytes due to spaces in the file were removed. Effectively, packing reduced the size of the input file by four times. In the unpacking, the number of data elements is read first. Then, the number of columns and that many of char strings are read. After that packed data is read byte by byte followed by dividing bytes into two nibbles such that first and last four bits extracted as integers. While packing we had to pay special attention to the case when both the number of columns and the number of rows are odd numbers. In this case the total number of the data elements will be odd and therefore not divisible by 2. In order to avoid this case, we remove the last row of the data before packing so that the number of elements is even.

3.4. Compression

To speed up the transmission of data from master processor to the slave processors, data is compressed losslessly and written to the disk space where it can be read by other processors. We noticed that there is strong correlation among neighboring elements in some sample data set even after packing two nibbles to a byte. Hence, we investigated further possibilities of compression with different techniques on the packed data. In particular, we experiment with Burrows–Wheeler transformation (BWT) [7], linear order transformation (LOT) [2], move-to-front (MTF) [5], inversion ranks [3] and a structured arithmetic coder [14]. Approximately 12–14 times better compression is achieved when first LOT or BWT is used, followed by inversion ranks and a structured arithmetic coder. When the underlying data elements are 8-bit, such as ASCII characters, LOT needs 256 additional integer indices (pointers). Yet, 8-bit data, such as ASCII characters, BWT requires $4n$ integer indices (pointers) where n are the size of the data string. Decompression is also done in a similar manner. Decoder gets the bytes and decodes them. When the number of bytes reaches to two mega bytes, it sends the data to inversion coder, and finally the data is sent to reverse LOT transformer.

After packing and compression, the original input file entering to the packing process is reduced by 40–75 times depending the file size and the repetitions of the values in the file. As an important point, this has to be noted that some of the above preprocessing steps including, fault addition, packing, and compression are optional hence not necessary for the main software to be executed.

3.5. Particle swarm optimization for searching the best BN

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 the solutions of the best performing particles in its neighborhood. At each optimization step, each particle adjusts its candidate solution (flies) according to the set of equations below.

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

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

Eq. (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

Table 1
List of variables used to evaluate the dynamical swarm response given in Eq. (10)

$\vec{v}_i$	The particle velocity
$\vec{x}_i$	The particle position (test solution)
t	Time
ϕ_1	A uniform random variable usually distributed over [0,2]
ϕ_2	A uniform random variable usually distributed over [0,2]
$\vec{x}_{i,p}$	The particle's position (previous) that resulted in the best fitness so far
$\vec{x}_{i,n}$	The neighborhood position that resulted in the best fitness so far

will result in a better fitness. Particle swarm optimization can be applied to any problem as long as the fitness function can be defined for the particles.

A simple list of ingredients is required to implement the PSO technique [26]: A particle construct that fully represents the solution; A fitness calculator; A dynamic response of the swarm, see Eq. (10).

The particle must represent a possible solution to the problem. In this application, each particle represents a BN as shown in Fig. 2. 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.

To calculate the fitness, we use the scoring of Herskovits [22] given by

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

where r_i is the number of states for node i, the first product is over the nodes in the network, the second product 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 Eq. (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 set of parents. Eq. (11) can be derived from Eq. (9); see [22] for a derivation.

At each step of the optimization, Eq. (10) is used to evaluate the particle velocities, but since our particle is binary [25], 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 Bayesian-scoring function given in Eq. (11) and our particles move with the dynamics of a binary particle swarm [25].

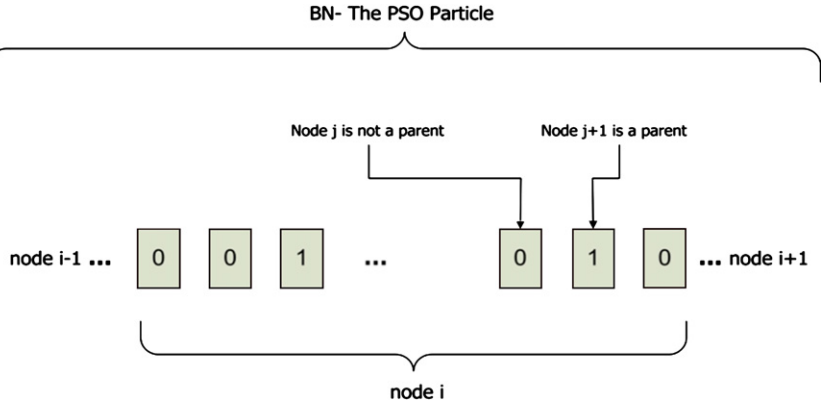

Fig. 2. The PSO particle representation of a Bayesian network.

3.6. Parallel computing for particle swarm optimization

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 in parallel. The pseudo-code of the PSO algorithm is shown below.

```
initialize the particle swarm
do until max steps reached
  for all particles
    send particle to slave
  end for
  receive all particle fitness calculations from slaves
  update particle bests and neighborhood bests
  move the particles
end do
```

For large numbers of variables and large datasets, the fitness calculation is the most computationally expensive aspect of the BN search. We therefore have implemented an MPI (message passing interface) program to exploit this inherent parallelism. We adopt a master–slave topology for our MPI implementation as illustrated in Fig. 3. This topology is adequate especially because of the synchronous aspect of the PSO algorithm, which will be discussed in the next section.

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 using MPI directives. Each slave process waits for a particle from the master; upon its receipt, it calculates its fitness and sends it back. Fig. 4 shows the ***main*** program that initializes the MPI and creates the master and the slaves. Figs. 5 and 6 show ***master*** and ***slave*** functions respectively. We show the MPI related and important parts of the program because of its large length.

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. Because the master must wait for all fitnesses to be returned by the slaves, we should, if possible, use a number of slave processes equal to the number of particles in the swarm. If we have fewer slave processes than particles, one or more slaves will need to calculate the fitness of more than one particle. Naturally, if the number of particles in the swarm is less than the number of participating processes, the extra processes will simply sit idle during the optimization. Fig. 7 presents the parallel implementation of the PSO algorithm by highlighting the MPI commands.

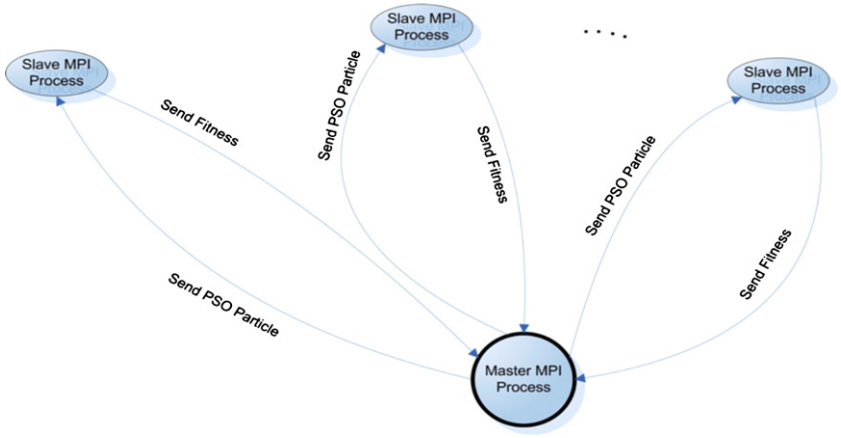

Fig. 3. Parallel implementation of PSO algorithm.

```
main( int argc, char  **argv) {

int myrank;

MPI_Init(&argc, &argv);    /* initialize MPI */

MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* process rank, 0 thru N-1 */
.
.
. Main function argument initializations
.
. Below code makes the decision of becoming the master or a slave based on the rank
.
if (myrank == 0 && argc==5) {
        master(cases_filename,i_cases,i_nodesizes,i_nodenames,pass_pso_steps,pass_neighsize,pass_NumParticles);

else if(argc==5) {
        slave(myrank,cases_filename);
}

MPI_Finalize();          /* cleanup MPI */


return 0

}
```

Fig. 4. Part of the code that initializes, handles, and cleans MPI, the master and the slave nodes.

```
int master(char* fn_case,int s_cases,int s_nodesizes,int s_nodenames,int pass_pso_steps,int pass_neighsize,int
pass_NumParticles)  {
.
. Various initialization statements
.
MPI_Status                status;
MPI_Comm_size(MPI_COMM_WORLD,   &ntasks);
.
. Master node is initialized with the data and particles are created and initialized
.
do
{
.
MPI_Send(&SendArray,14999,MPI_INT,rank,iParticle,MPI_COMM_WORLD);
.
.Here particles are sent to the slaves one by one
.
}while (last particle);

While(not last particle)
{
.
MPI_Recv(&result,1,MPI_DOUBLE,MPI_ANY_SOURCE,MPI_ANY_TAG,MPI_COMM_WORLD,&status);
.
. Here fitness results are received from the slaves
.
}
.
.After  number of PSO step is reached, the following code piece is run to kill all the slaves
.
for(int k=1;k<ntasks;k++)          {
        MPI_Send(0,0,MPI_INT,k,DIETAG,MPI_COMM_WORLD);
}
.
. Statements for writing results to files
.
return 0;
}
```

Fig. 5. Function for the master node (MPI commands are red (in the web version), bold, and italic).

```
void slave(int myrank,char* cases_filename) {
.
.Statements for initialization of the slaves
.
. The following infinite loop is waiting particles from the master.
for (;;) {
        MPI_Recv(&work, 14999, MPI_INT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);

        . The following if statement ends the slave function when DIE signal comes from the master.
        if (status.MPI_TAG == DIETAG) {
        return;
        }
        .
        . The part of the code calculates the fitness based on the particle (Bayesian Network) and the data.
        .
        .The following MPI statement sends the fitness back to the master
        MPI_Send(&result, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
        } //End of the infinite loop

}; //End of the slave
```

Fig. 6. Function for the slave nodes (MPI commands are red (in the web version), bold, and italic).

As can be seen from Fig. 7, the parallel implementation of PSO algorithm is achieved by sending the particles to computing nodes (slaves). This enables the PSO algorithm to have all the fitnesses of the particles calculated at the same time and reduces the amount of time spent in the fitness calculation step by p times, where p is the number of particles. There are other approaches to implement parallel PSO based on the particle distribution and synchronization of the MPI topology. These approaches and complexity analysis are presented in the following section where we analyze the performance of the parallel implementation of the PSO algorithm.

3.6.1. Performance analysis for the parallel implementation of PSO

The performance of the parallel implementation of PSO can be analyzed by evaluating complexity of the PSO algorithm and the topology of the parallel implementation.

3.6.1.1. Evaluation of parallel PSO algorithm complexity. If the complexity of the structural Bayesian network learning algorithm is viewed as the number of networks that must be evaluated over the run of the algorithm, then the complexity of the PSO algorithm can be written as $O(kp)$ where k is the number of steps in the algorithm and p is the number of particles in the swarm. Although one can say that the complexity of the PSO algorithm is 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, represented as $k(n,p)$. This is mainly because the complexity of the search space and the number of possible network will exponentially increase if the number of nodes in a Bayesian network is increased. Thus, a more exact expression of the complexity of the PSO algorithm for the structural BN learning would be $O(k(n,p) \cdot p)$, where n is the number of nodes in the network, p is the number of particle in the PSO algorithm, and k is the number of steps in the PSO algorithm. Thus, one can say that the number of steps required for the PSO algorithm to converge depends on the number of nodes in the Bayesian network and the number of particles in the PSO algorithm. 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. As can be seen from the complexity representation, the relationship in $k(n,p)$ is data dependent and experiments should be run for each data set. Thus, it is not the subject of this paper. In this paper, we are trying to reduce the time consumed to go through each step in the PSO algorithm. As stated above, in the ith step of the PSO algorithm (k_i), there are p numbers of fitness calculations.

Therefore, by parallel implementation of the fitness calculations (all the particle fitnesses are calculated at the same time on their specific slave machines), the time spent for the fitness calculations will be reduced approximately by p times. This is an approximate because it assumes that time wasted on sending the particles

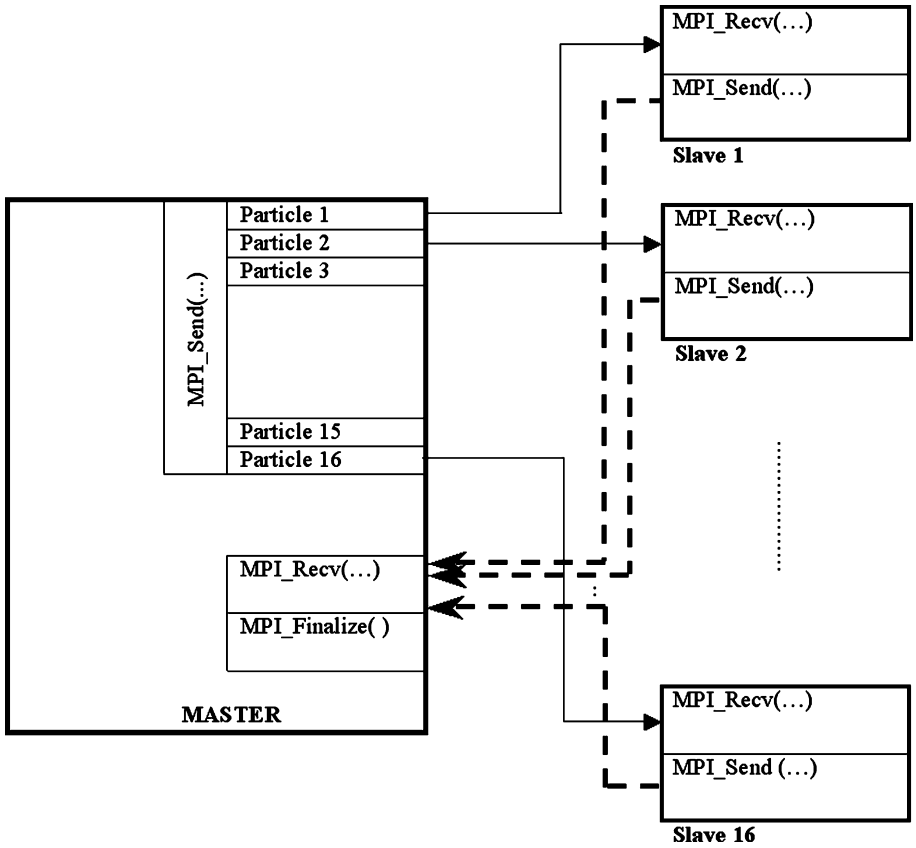

Fig. 7. MPI topology of the parallel implementation of PSO algorithm.

and receiving the fitness results is negligible compare to the amount of time it takes to calculate the fitness of a particle. The connection between the master and the slaves is achieved by a GigaBit intranet and a fitness calculation for an engine network takes about 1–20 s depending on the data length. Assuming each particle is n bits long, $p \cdot n$ bits will be transferred to send all the particles to the slaves. In addition, master receives fitness values from the slaves. The fitness of a particle is represented by a double in the program. Thus, the master receives $d \cdot p$ bits total where d is the number of bits needed to represent a double. In addition to the data transferred between slaves and the master, h bits used for the header of the packets transferred between the master and the slaves. Thus, the amount of time spent on transferring particles and fitnesses between master and slaves is $p(n + d + 2h)/r$ where r is the speed of the network in bits per second. Assuming p is 16, d is 32, h is 64 bits (worst-case estimate), n is 15 000 and r is 1 Gigabits per second, the amount of time spent on communication is 242.5 μs, which is negligible compare to 1–20 s of fitness calculation.

In addition to the time spent of transferring particles and fitnesses, the system spends considerable amount of time when the master sends the data to all the slaves at the initialization of the MPI. So, the system spends p times more time on loading the data for the PSO algorithm. However, compare to the reduction on the fitness calculation time, the time spent when loading the data to the slaves is negligible. Based on this basic analysis, one can claim that the parallel implementation of the PSO algorithm is approximately p times faster than the regular PSO algorithm. Thus, when the number of particles increases, the performance of the parallel PSO algorithm also increases. In our experiments we used 16, 32, and 45 particles in which the parallel PSO algorithm was approximately 16, 32, and 45 times faster than the regular PSO algorithm.

3.6.1.2. Evaluation of the topologies for the parallel PSO implementation. We have explored two topologies for the MPI implementation: ***synchronous*** and ***asynchronous***. In the synchronous approach, the master sends the particles to the slaves ***at the same time*** and waits for the arrival of fitnesses from ***all*** the slaves before it assigns

the new particle and neighborhood positions. As long as the cluster has more processes than the number of particles in the PSO algorithm and the time of completion of the fitness calculation in a slave 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 fitnesses. When there are adequate processes and the complexity of the fitness calculation is constant (mainly because of the uniform hardware and software properties of the slaves), all slaves will return particle fitnesses at approximately the same time. This will result in a small idle processing time. The maximum idle processing 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.

When the number of particles exceeds the number of processes in the cluster, and is not an even multiple of the number of processes, there will be a round of fitness calculation requests sent to the slaves where there will be fewer fitness calculations than processes. During this phase of the algorithm, it is possible, in the worst case, to have all but one processes become idle. This is clearly an undesirable use of the cluster. A possible solution is to implement asynchronous PSO.

In an asynchronous PSO implementation on a computer cluster, one changes the swarm dynamics and allows any single particle to execute its motion as soon as it has received an update on its fitness at its current position from a slave process. This method of handling of the particle swarm will ensure that every slave process is busy calculating a fitness value at any given time. Another desirable property of this implementation is that we no longer care if the complexity of the particle fitness calculations is constant across particles, at least in so far as the complexity of the calculation impacts the cluster efficiency. Since we have large number of nodes in our computer cluster and the number of particles is a complexity parameter in our software, we implemented the parallel PSO algorithm as a synchronously evolving particle swarm.

3.6.2. Particle 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. Having set 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 restrict the maximum number of arcs to $2n$, where n is the number of nodes. This restriction has no impact on the particle initialization since we initialize with $N/2$ edges. We initialize each component of each particle's velocity randomly on the interval

$$-v_{\max} \leqslant v_0 \leqslant v_{\max} \tag{13}$$

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 have tested the performance of the algorithm for maximum velocities of 6, 8, and 10.

3.7. The software

There are commercial Bayesian network software packages such as Hugin [56], GeNIe [59] and Netica [57], WinMine [55] and others [54]. Most of these software packages do not have structural Bayesian network learning. Our software is not commercially available but can be requested from the authors. Our BN software can be executed in two modes; simulation and inference. Simulation mode generates BN from the data generated by the preprocessing (training data). The parameters required for simulation mode are number of steps, name of the input file, number of PSO particles and network neighborhood. Network neighborhood determines the number of fitness scores (of particles) each particle is allowed to evaluate. The file name is the name of the input file produced from raw data by preprocessing and compressing. Inference mode requires BN generated through the simulation mode, the data file used to generate this BN, and the inference file to test against the BN. All of the data files must be preprocessed, packed, and compressed. Preprocessing is done by using MATLAB programs while packing and compression are done with C/C++ routines. All of these steps (preprocessing, packing, and compression) are implemented in windows environment on a PC with the following

specifications; Pentium IV, 3.2 GHz, 2 GB memory. At the end of the compression step what we have is just one small compressed file to be fed to the BN software which is written in C/C++ with embedded MPI (MPICH 1.2.6 – available on the web [60]) commands on Linux environment. Thus, the decompression algorithm is implemented in C/C++ in Linux. The software is executed on computer cluster with 24 boxes where each box has two CPUs (AMD Opteron 1.8 GHz) and 2 GB of memory.

4. Experimental results

We study the performance of the algorithm in two aspects: performance of fault diagnosis and performance of training. The performance of the training (simulation mode) is based on the performance of particle swarm optimization (PSO) algorithm implemented on a computer cluster in order to discover the best Bayesian network that fits the data best. The performance of PSO is analyzed with respect to the number of particles, the number of PSO steps, the maximum velocity of the particles, and the neighborhood size. We studied the number of PSO steps and the neighborhood size and found that the performance of the PSO algorithm does not improve with the PSO steps larger than 3000. Thus, we have chosen the number of PSO steps as 3000 for all the runs. Our studies show that the neighborhood size of 0 (global neighborhood) gives the best convergence rate and results in better networks.

We use an index-based neighborhood. 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. For example 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 size of zero implies a global neighborhood where all particles can see fitness values of all other particles in the swarm; there is one global neighborhood. In this work, we study performance of the algorithm with respect to the maximum velocity of the particles and the number of particles. We have set the maximum particle velocity to 6, 8, and 10. In addition, we have run simulations with 16, 32, and 45 particles in the PSO. Thus in the simulation mode, we have run the PSO algorithm with none different parameter sets. For each set, we have performed five runs (total of 45 runs).

The performance of fault diagnosis is related to inference mode where a data file obtained from an engine is tested using the Bayesian network discovered in the training (simulation) mode. These test files have 15 oil related sensor readings. Thus, the inference reads each line of the file and calculates the probability of fault given the values in this line. The probabilities of fault are calculated for each line in the file and their average is calculated. If the average probability of the fault is higher than a set threshold, the file is marked as faulty. In this work, we have run inference using all the valid networks (fault node has a parent) obtained from the runs performed in the simulation mode. Then, we present the performance of the runs with respect to successful fault diagnosis.

4.1. Performance of the Bayesian network discovery with PSO algorithm

We study two parameters of the algorithm: the maximum particle velocity and the neighborhood size. In addition, we study the performance of our new conditional probability table calculation scheme, called memorizing CPT. We have performed runs for maximum velocities of 6, 8, and 10 with particle sizes of 16, 32, and 45. Fig. 8 illustrates the fitness values of the Bayesian network discovered in each run.

As can be seen in Fig. 8, maximum velocity of the particles and the number of particles determines the convergence and the fitness of the network. Note that the lower fitness values are better since the sign of the fitness is negative. As the maximum velocity of the particles increases, the average fitness increases. Thus, increasing the velocity of the particles does not really help the algorithm. The reason is the particles move faster and it is hard for them to settle in local optima. Based on the simulation results, there is no big difference between maximum velocities 6 and 8 in terms of fitness. However, since the maximum velocity of 8 gave the best fitness values regardless of the number of particles in the PSO we chose to use maximum velocity of 8 in our concluding simulations. When we look at the performance of the PSO with respect to the number of particles, we can say that having 32 particles gives the best performance. Thus, we chose to have 32 particles in the PSO

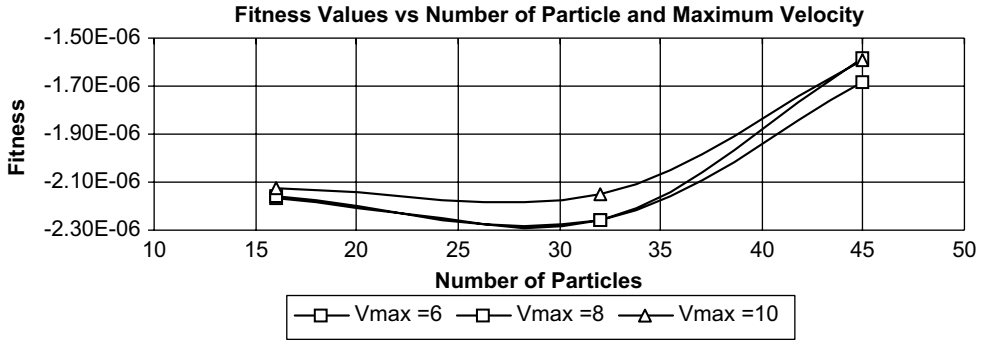

Fig. 8. Performance of the PSO Algorithm based on velocity and swarm size.

algorithm. We can conclude that maximum velocity of 8 and 32 particles are the best parameters for the PSO algorithm.

4.2. Memorizing scheme for conditional probability table calculations

In addition to discovering the optimal maximum particle velocity and the neighborhood size, we also improved the calculation of the conditional probability tables (CPTs). The CPT table entries are calculated based on the candidate network's structure and the available data. When there are more than six parents to a node in the network, the CPT of the node has more than 16 184 elements ($4 * 4^6$). In the PSO algorithm, the networks are generated based on the PSO dynamics and similar networks will be explored in consecutive PSO steps. Thus, for some nodes the same CPTs will be calculated. We designed a memorizing scheme where the CPT tables are stored in hash tables when they are calculated for the first time. In the following steps, if the same CPT calculation is required for the score calculation, we pull the stored CPT from the hash table instead of calculating the CPT again. In initial steps of the algorithm, the hash table is filled with the CPTs. When the PSO algorithm starts to converge more and more CPTs are repeated in consecutive networks. Thus, at the end of the PSO, the number of required CPT calculations reduces significantly. Each slave process has its own hash table instead of having a large hash table in the master process in order to reduce the network traffic in the cluster. Because of the memorizing scheme, the PSO steps take less time as the algorithm starts using the memorized CPTs. We have run 10 simulations with and without memorizing scheme with 16 particles, global neighborhood, and 2000 PSO steps. On average, the PSO runs without memorizing scheme took about 25.6 h. On the other hand, the same simulations took about 3.5 h on average when we use the memorizing scheme. Thus, the memorizing scheme increases the convergence rate about 7.3 times.

4.3. Fault prediction for new datasets – inference mode

Once a BN of the system is constructed using a complete dataset, it will serve as a tool for fault prediction of new datasets. Having these new datasets (test files) and the BN at hand, we are now required to infer about the health of the engine. We run our BN software in inference mode by using previously constructed BN and new data files [45]. The algorithm calculates the probability of the engine reading being faulty by calculating the average of the probability of fault for each line (sensor readings at time t). In Table 2, we present inference results of the five networks that gave the best fault detection.

As seen in Table 2, the performance of the inference provides good fault detection since all five networks can differentiate faulty and non-faulty files successfully. As mentioned before, we have run a total of 45 simulations. Eleven of these runs resulted in networks where the fault variable had no parents. In this case, no useful inference can be performed for fault diagnosis. The simulation results show that we get better inference results when the number of parents of the fault variable is high. However, having high number of parents does not always give good results. It is also important that which variables are the parents of the fault variable. Fig. 9 shows the inference performance (fault diagnosis performance) with respect to the number of parents

Table 2
Fault detection percentages of five best inference results

V_{max}	Particles	% Fault for faulty files	% Fault for non-faulty files	Parents of fault node
8	32	91.1	45.18	3
10	16	90.02	50.67	3
6	45	89.85	51.52	3
6	45	89.1	55.33	2
10	45	89.1	55.33	2

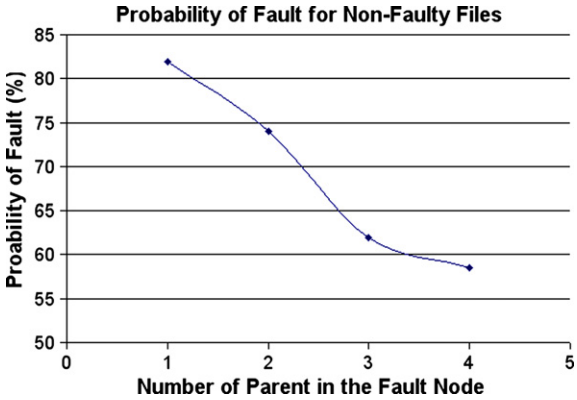

Fig. 9. The effect of the number of parent in the Fault node in fault diagnosis.

the fault node has. We have run inference on non-fault data and determined the number of parents of the fault variable. Then, we take the average of the inference results of the networks, which has same number of parents going into the fault variable. As seen in Fig. 9, the fault prediction ability of the BN improves, on average, when the fault node has more parents.

Based on the simulations and the fitness values of the resulting networks, we concluded that during the PSO search some of the nodes assigned high number of parents resulting in a fitness increase. This fitness increase causes the PSO algorithm to get stuck in a local minimum. Thus, the Fault node gets less number of parents. In order to increase the possibility of having large number of parents in the fault variable, we have limited the number of parents a node can have to six. We have run five simulations with 16 particles, the maximum velocity of eight, and 2000 PSO steps. The fitness and the fault probabilities for these runs are presented in Table 3.

Table 3
Inference results of Bayesian networks with parent size limitation

Fitness score (e-06)	Test file for inference	Inference result (fault %)
−2.151036	Not faulty	70.175011
	Faulty	86.219025
−1.948312	Not faulty	75.345055
	Faulty	85.127258
−2.233262	**Not faulty**	**28.773972**
	Faulty	**94.335938**
−2.043306	Not faulty	79.860886
	Faulty	84.633240
−2.164772	Not faulty	58.197964
	Faulty	63.137062

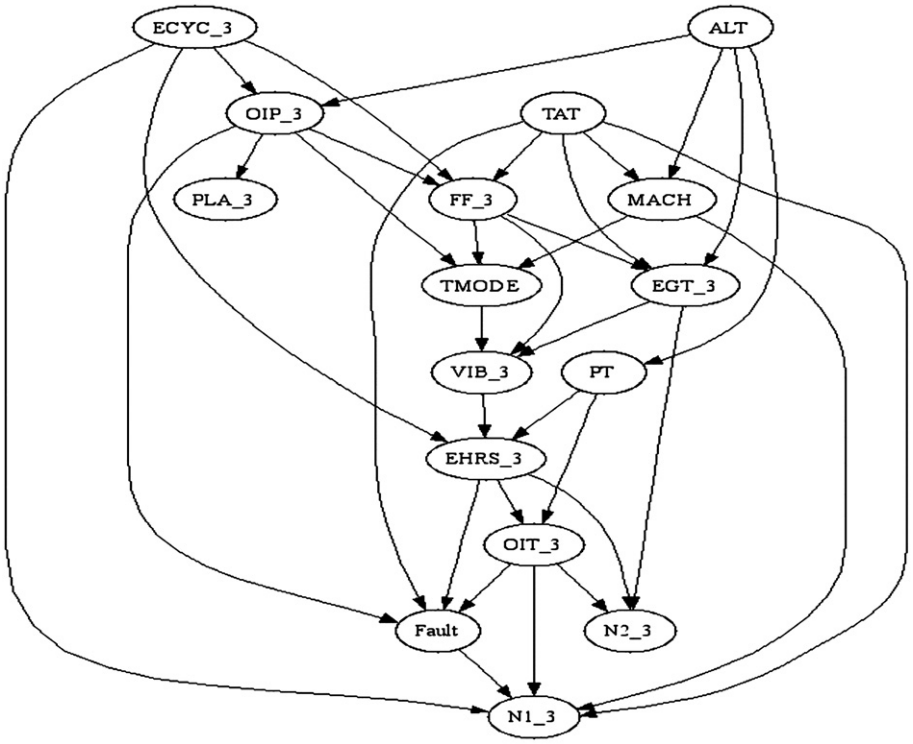

Fig. 10. Bayesian network with the best inference BN generated by our software.

In Table 3, the fitness score represents how well the training data fits the resulting BN. Inference results are shown as the average probability of fault for the inference files (test files). The networks with smaller fitness scores tend to distinguish faulty and not faulty files better. However this may not always hold as fitness scores relate the training data to the discovered BN while the fault probability relates to the sub-network that includes the fault variable (i.e., fault variable and its parents). As can be seen in Table 3, one of the BN shows excellent inference results since the fault variable has four parents. Fig. 10 presents this Bayesian network drawn by a graph visualization program called GraphViz [58].

5. Conclusions

In this paper, we have successfully implemented a fault diagnosis technique for airplane engines using the particle swarm optimization (PSO) algorithm for learning the structure of a BN from a large dataset. The PSO algorithm is parallel in nature: all fitness calculations can be performed in parallel in each step. We have exploited this feature and implemented our optimization on a computer cluster consisting of 48 CPUs. Parallel computation of fitnesses allows us to perform the network extraction in a fraction of time, scaling linearly with the number of processes in the cluster, which would be required on a single thread platform. The advantages of our approach are that we make no assumptions about the ordering, independence, or the leaf node attributes of variables in the BN. There is no expert knowledge (i.e. order of the variables) necessary about the data unlike other BN-based fault diagnosis techniques. Our algorithm can also handle large systems (domains with more than 25 variables are considered large in the literature [29]). Once the BN is constructed from data, it can be used in any decision support system either online or offline. Online uses could include monitoring the system and diagnosing failures within the system (or engine in our case). Offline uses include predicting fault and understanding the system. These uses can help airplane companies save money by cutting maintenance costs (i.e., reduce the number of unnecessary replacement of parts) thus prevent cancellation or delay of flight due to engine faults and reduce the time required to detect fault. One of the other nice features of our technique is its

ease of modifiability, for example, even though we added one fault variable to diagnose faults in engines, several fault variables with binary states or one fault variable with three (or more) states could be added to diagnose different types of faults. The results show that a Bayesian network can be learned from engine data and successful inference can be performed to detect the anomalies or faults in the sensor readings of an airplane engine.

References

[1] S. Acid, L.M. de Campos, Searching for Bayesian network structures in the space of restricted acyclic partially directed graphs, Journal of Artificial Intelligence Research (2003) 445–490.
[2] Z. Arnavut, Lossless compression of pseudo-color images, Optical Engineering 38 (1999) 1001–1005.
[3] Z. Arnavut, Inversion coder, The Computer Journal 47 (2004) 46–57.
[4] I.A. Beinlich, H.J. Suermondt, R.M. Chavez, 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 38 (1989) 247–256.
[5] J.L. Bentley, D. Sleator, R.E. Tarjan, V.K. Wei, I. Munro, A locally adaptive data compression scheme, Communications of the ACM 29 (1986) 320–330.
[6] R. Blanco, I. Inza, P. Larrañaga, Learning Bayesian networks in the space of structures by estimation of distribution algorithms, International Journal of Intelligent Systems 18 (2003) 205–220.
[7] M. Burrows, D.J. Wheeler, A Block-sorting Lossless Data Compression Algorithm, Digital Systems Research Center, Palo Alto, California, 1994.
[8] D.M. Chickering, D. Gieger, D.E. Heckerman, Learning Bayesian Network is NP-hard, Microsoft Research, 1994.
[9] D.M. Chickering, Learning equivalence classes of Bayesian-network structures, Journal of Machine Learning Research 2 (2002) 445–498.
[10] D.M. Chickering, Optimal structure identification with greedy search, Journal of Machine Learning Research 3 (2002) 507–554.
[11] G.F. Cooper, E. Herskovits, A Bayesian method for constructing Bayesian belief networks from databases, Proceedings of the Conference on Uncertainty in AI (1990) 86–94.
[12] G.F. Cooper, E. Herskovits, A Bayesian method for the induction of probabilistic networks from data, Machine Learning (Historical Archive) 9 (1992) 309–347.
[13] L.M. de Campos, J.M. Fernandez-Luna, J.A. Gamez, J.M. Puerta, Ant colony optimization for learning Bayesian networks, International Journal of Approximate Reasoning 31 (2002) 291–311.
[14] P.M. Fenwick, The Burrows–Wheeler transform for block sorting text compression: principles and improvements, The Computer Journal 39 (1996) 731–740.
[15] N. Friedman, Learning belief networks in the presence of missing values and hidden variables, Proceedings of Fourteenth International Conference on Machine Learning (1997) 125–133.
[16] N. Friedman, I. Nachman, D. Peer, Learning Bayesian network structure from massive datasets: the "sparse candidate" algorithm, in: Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence, 1999.
[17] N. Friedman, D. Koller, Being Bayesian about Network structure, in: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, 2000.
[18] N. Friedman, D. Koller, Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks, Machine Learning 50 (2003) 95–125.
[19] D. Heckerman, D. Gieger, M. Chickering, Learning Bayesian networks: the combination of knowledge and statistical data, Microsoft Research, Redmond Washington, Technical Report, 1994.
[20] D. Heckerman, A tutorial on learning Bayesian networks, Microsoft Research, Redmond Washington, Technical Report, 1995.
[21] D. Heckerman, C. Meek, G. Cooper, A Bayesian approach to casual discovery, Microsoft Research, Redmond Washington, Technical Report, 1997.
[22] E. Herskovits, Computer-based Probabilistic Network Construction, Medical Informatics, Stanford University, 1991.
[23] F.V. Jensen, An Introduction to Bayesian Networks, UCL Press, London, 1996.
[24] M.J. Kane, A. Savakis, Bayesian network structure learning and inference in indoor vs. outdoor image classification, Proceedings of the 17th International Conference on Pattern Recognition 2 (2004) 479–482.
[25] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm algorithm, Proceedings of the IEEE Conference on Systems, Man, and Cybernetics (1997) 104–4109.
[26] J. Kennedy, R.C. Eberhart, Swarm Intelligence, Morgan Kaufman Publishers, San Francisco, 2001.
[27] O. Kipersztok, G.A. Dildy, Evidence-based Bayesian networks approach to airplane maintenance, 2002.
[28] H. Kirsch, K. Kroschel, Applying Bayesian networks to fault diagnosis, in: Proceedings of the 3rd IEEE Conference on Control Applications, 1994, pp. 895–900.
[29] M. Koivisto, K. Sood, Exact Bayesian structure discovery in Bayesian networks, Journal of Machine Learning Research 5 (2004) 549–573.
[30] W. Lam, F. Bacchus, Learning Bayesian belief networks: an approach based on the MDL principle, Computational Intelligence 10 (1994) 269–293.

[31] P. Larranaga, M. Poza, Y. Yurramendi, R.H. Murga, C.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 (1996) 912–926.
[32] D. Madigan, J. York, D. Allard, Bayesian graphical models for discrete data, International Statistical Review (Revue internationale de statistique) 63 (1995) 215–232.
[33] T.A. Mast, A.T. Reed, S. Yurkovich, M. Ashby, S. Adibhatla, Bayesian belief networks for fault identification in aircraft gas turbine engines, Proceedings of the IEEE International Conference on Control Applications 1 (1999) 39–44.
[34] C. Meek, Graphical Models, Selecting Causal and Statistical Models, Ph.D. Dissertation, Carnegie Mellon University, Pittsburg, 1997.
[35] N. Mehranbod, M. Soroush, M. Piovoso, B.A. Ogunnaike, Sensor fault detection and identification via Bayesian belief networks, Proceedings of the American Control Conference 6 (2003) 4863–4868.
[36] J.W. Myers, K.B. Laskey, K.A. DeJong, Learning Bayesian networks from incomplete data using evolutionary algorithms, in: Proceedings of the International Conference on Genetic Algorithms (GECCO-99), 1999.
[37] D. Nikovski, Constructing Bayesian networks for medical diagnosis from incomplete and partially correct statistics, IEEE Transactions on Knowledge and Data Engineering 12 (2000) 509–516.
[38] K.G. Olesen, S.L. Lauritzen, F.V. Jensen, A HUGIN: a system for creating adaptive causal probabilistic networks, in: Proceedings of the Eighth Conference on Uncertainty in AI (UAI '92), 1992, pp. 223–229.
[39] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufman, San Francisco, 1988.
[40] H. Peng, C. Ding, Structure search and stability enhancement of Bayesian networks, in: Proceedings of 3rd IEEE International Conference on Data Mining, 2003, pp. 621–624.
[41] K.W. Przytula, D. Thompson, Construction of Bayesian networks for diagnostics, Proceedings of the IEEE Aerospace Conference 5 (2000) 193–200.
[42] M. Ramoni, P. Sebastiani, Robust learning with missing data, Machine Learning 45 (2001) 147.
[43] R.W. Robinson, Counting unlabeled acyclic digraphs, in: C.H.C. Little (Ed.), Combinatorial Mathematics V, Lectures Notes in Mathematics, vol. 622, Springer-Verlag, New York, 1977, pp. 28–43.
[44] S. Russell, J. Binder, D. Koller, Adaptive probabilistic networks, Technical Report UCB/CSD-94-824, 1994.
[45] F. Sahin, A Bayesian Approach to the Self-Organization and Learning in Intelligent Agents, Ph.D. Dissertation, Virginia Polytechnic Institute and State University, Blacksburg,Virginia, 2000.
[46] G. Schwarz, Estimation the dimension of a model, Annals of Statistics 6 (1978) 462–464.
[47] Sebastiani Paola, Sebastiani Paola, Ramoni Marco, Bayesian inference with missing data using bound and collapse, Journal of Computational & Graphical Statistics 9 (2000).
[48] D.J. Spiegelhalter, P. Dawid, S.L. Lauritzen, R. Cowell, Bayesian analysis in expert systems, Statistical Science 8 (1993) 219–282.
[49] D.J. Spiegelhalter, R.G. Cowell, Learning in probabilistic expert systems, Bayesian Statistics 4 (1992).
[50] O. Uluyol, K. Kim, C. Ball, On-board characterization of engine dynamics for health monitoring and control, in: Proceedings of GT2005, ASME Turbo Expo 2005: Power for Land, Sea and Air, Reno-Tahoe, Nevada, 2005.
[51] F. Wittig, A. Jameson, Exploiting qualitative knowledge in the learning of conditional probabilities of Bayesian networks, in: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, 2000.
[52] M.C. Yavuz, F. Sahin, Z. Arnavut, O. Uluyol, Generating and exploiting Bayesian networks for fault diagnosis in airplane engines, in: Proceedings of the IEEE International Conference on Granular Computing, 2006, pp. 250–255.
[53] Z. Yi-feng, P. Kim-leng, Block learning Bayesian network structure from data, in: Proceedings of the 4th International Conference on Hybrid Systems, 2004, pp. 14–19.
[54] BN Tools. <http://www.cs.ubc.ca/~murphyk/Software/BNT/bnsoft.html>.
[55] Microsoft WinMine. <http://research.microsoft.com/~dmax/WinMine/tooldoc.htm>.
[56] Hugin. <http://www.hugin.com>.
[57] Netica. <http://www.norsys.com/networklibrary.html>.
[58] GraphViz. <http://www.graphviz.org>.
[59] Genie. <http://www2.sis.pitt.edu/~genie/>.
[60] MPICH. <http://www-unix.mcs.anl.gov/mpi/mpich/>.