Learning Bayesian Classifiers using Overlapping Swarm Intelligence

Nathan Fortier
Compute Science Department
Montana State University
Bozeman, Montana 59714
Email: nathan.fortier@montana.edu

John Sheppard
Compute Science Department
Montana State University
Bozeman, Montana 59714
Email: john.sheppard@cs.montana.edu

Shane Strasser
Compute Science Department
Montana State University
Bozeman, Montana 59714
Email: shane.strasser@montana.edu

***Abstract*—Bayesian networks are powerful probabilistic models that have been applied to a variety of tasks. When applied to classification problems, Bayesian networks have shown competitive performance when compared to other state-of-the-art classifiers. However, structure learning of Bayesian networks has been shown to be NP-Hard. In this paper, we propose a novel approximation algorithm for learning Bayesian network classifiers based on Overlapping Swarm Intelligence. In our approach a swarm is associated with each attribute in the data. Each swarm learns the edges for its associated attribute node and swarms that learn conflicting structures compete for inclusion in the final network structure. Our results indicate that, in many cases, Overlapping Swarm Intelligence significantly outperforms competing approaches, including traditional particle swarm optimization.**

I. Introduction

Bayesian networks have proven to be a useful tool for reasoning under uncertainty and have been applied in a variety of fields. For instance, Bayesian networks have been used in the field of bioinformatics to identify gene regulatory networks [1]. They have also been applied to system diagnostics as a method for predicting failures and tracking degradation [2], [3], [4]. Many tasks that utilize Bayesian networks can be viewed as classification problems. These problems require the identification of a class label for instances described by a set of attributes. Many authors have used Bayesian networks for classification in medical diagnostics, music identification, and other areas [5], [6], [7].

However, using Bayesian networks for classification requires the construction of an effective model. Learning a Bayesian network model from data has been shown to be NP-Hard [8] and several authors have applied machine learning techniques to the problem [9], [10], [11]. In the area of classification, model learning typically consists of learning relationships between the various features in the classification problem. Learning classification models from pre-classified data is an active research topic in machine learning.

We propose a swarm based approximation algorithm for the problem of Bayesian classifier learning. Our approach is based on the overlapping swarm intelligence (OSI) framework, first introduced in [12]. In our algorithm, a swarm is associated with each attribute in the data to be classified. Each swarm learns the parent/child relationships for its corresponding attribute. Swarms that learn conflicting parent/child relationships compete for inclusion in the final network structure.

II. Background

A. Bayesian Networks

A Bayesian network is a directed acyclic graph that encodes a joint probability distribution over a set of random variables, where each variable can assume one of an arbitrary number of mutually exclusive values [13]. In a Bayesian network, each random variable is represented by a node, and edges between nodes in the network represent probabilistic relationships between the random variables. Each root node contains a prior probability distribution while each non-root node contains a probability distribution conditioned on the node's parents. For the set of random variables in the network, the probability of any entry of the joint distribution can be computed using the chain rule.

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i | X_{i+1}, ..., X_n)$$

Using the local distributions specified by the BN, the joint distribution can be represented equivalently as

$$P(X_1, \dots, X_n) = \prod_{i=1}^{n} P(X_i | \text{Pa}(X_i)).$$

where $\text{Pa}(X_i)$ denotes the parents of X_i.

1) Structure Learning in Bayesian Networks: While, in simple cases, a Bayesian network can be developed by an expert, in many cases the problem of defining a Bayesian network is too complex for manual construction. In these cases Bayesian network structure and parameters can be learned from data. One type of method for general Bayesian network structure learning is score-based search. Score-based methods rely on a function to evaluate how well the network model matches the data, and they search for a structure that maximizes this function. A common scoring method to evaluate Bayesian network structures is by the probability of the data given the model. It was shown in [14] that the probability of the data D given a candidate Bayesian network structure B can be computed as follows.

Let $\mathbf{X}$ be the set of n discrete variables in B, where each variable $X_i \in \mathbf{X}$ has r_i possible value assignments: $(x_1, ..., x_{r_i})$. Let D be a set of data containing m cases, where each case contains a value assignment for each variable in $\mathbf{X}$. Each variable $X_i \in B$ has a set of parents, represented by $\text{Pa}(X_i)$. Let w_{ij} denote the j^{th} unique instantiation of

Pa(X_i) relative to D. Define q_i to be the number of unique instantiation of Pa(X_i) relative to D. Let N_{ijk} be the number of cases in D in which variable X_i has the value x_k and Pa(X_i) is instantiated as w_{ij}. Let

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

Then

$$P(D|B) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!. \tag{1}$$

Since the values for $P(D|B)$ can be very small, the log likelihood of the data given the model is often used as a scoring metric:

$$L(D|B) = \log P(D|B). \tag{2}$$

Another scoring metric commonly used for Bayesian network model selection is the Bayesian Information Criterion (BIC) developed by Gideon E. Schwarz [15]. This metric introduces a penalty term for the number of parameters in the candidate model. The formula for the BIC [16] is as follows:

$$\text{BIC} = -2 \log P(D|B) + p \log(m) \tag{3}$$

where p is the number of free parameters.

In the area of classification a less general type of Bayesian network structure is often learned. Given a classification problem consisting of a series of attributes A_i and a class label C, a Bayesian classifier is a Bayesian network in which a node is associated with a each attribute A_i and the class C. Typically, an edge is placed in the structure from the class node to each of the attribute nodes. Classification is performed by computing the most probable state of the class C given the states of the attributes $A_1, ..., A_n$. One approach to Bayesian classification is the Naive Bayes classifier. In this structure all attributes are assumed to be conditionally independent given the class, and there are no edges connecting any of the attribute nodes.

An alternative to Naive Bayes that relaxes the independence assumption is the Tree Augmented Naive Bayes (TAN) classifier described in [17]. The TAN algorithm strives to add edges to the network that maximize the conditional mutual information between the connected feature nodes given the class, defined as follows:

$$I(X, Y|C) = \sum_{x,y,c} P(x, y, c) \log \frac{P(x, y|c)}{P(x|c)P(y|c)} \tag{4}$$

where X and Y are the feature variables, and x, y, and c are the states of variables X, Y, and C respectively. This function computes the information that Y provides about X when the state of the class variable C is known. The TAN algorithm is described as follows:

1) Construct an undirected graph G containing a node for each feature.
2) Place a weighted edge between each pair of nodes X and Y where the weight of the edge is defined as $I(X, Y|C)$.
3) Find the maximum weighted spanning tree T of the graph G.
4) Add direction to the edges of T by selecting a root variable and setting the direction of all edges to be outward from it.
5) Create a class node C and add an edge from C to each feature node.

The TAN algorithm has been shown to have higher accuracy that Naive Bayes on many problems.

Algorithm 1 Particle Swarm Optimization

```
repeat
  for each particle position $x_i \in \mathbf{P}$ do
    Evaluate position fitness $f(x_i)$
    if $f(x_i) > f(p_i)$ then
      $p_i = x_i$
    end if
    if $f(x_i) > f(p_g)$ then
      $p_g = x_i$
    end if
    $v_i = \omega v_i + \text{U}(0, \phi_1) \otimes (p_i - x_i) + \text{U}(0, \phi_2) \otimes (p_g - x_i)$
    $x_i = x_i + v_i$
  end for
until termination criterion is met
```

B. *Particle Swarm Optimization*

The Particle Swarm Optimization (PSO) algorithm was first proposed by Eberhart and Kennedy [18]. PSO is a population based search technique inspired by the behavior of fish schools and bird flocks. In PSO the population is initialized with a number of random solutions called particles. Each particle has a position that encodes a potential solution in the search space and a velocity that defines how the particles will move through the search space. Each particle keeps track of the coordinates in the search space that are associated with the best solution it has found so far. These coordinates are called the personal best position of the particle and denoted p_i for the ith particle. The algorithm also keeps track of the overall best solution value and location, found so far by any particle in the population. This is called the global best position and is denoted p_g.

Both position and velocity are typically defined as vectors of real numbers. The search process updates the position vector of each particle based on that particle's corresponding velocity vector. These velocity vectors are updated at each iteration based on the fitness of the states visited by the particles. Eventually all particles move closer to an optimum in the search space. The pseudocode for the traditional PSO algorithm is presented in Algorithm 1.

In Algorithm 1, $\mathbf{P}$ is the particle swarm, U(0, ϕ_i) is a vector of random numbers uniformly distributed in [0, ϕ_i], $\otimes$ is component-wise multiplication, v_i is the velocity of a particle, and x_i is the object parameters or position of a particle. Three parameters need to be defined for the PSO algorithm:

- ϕ_1 determines the maximum force with which a particle is pulled toward p_i;
- ϕ_2 determines the maximum force with which a particle is pulled toward p_g;
- ω is the inertia weight.

The inertia weight ω is used to control the scope of the search and eliminate the need for specifying a maximum velocity. Even so, it is customary to specify maximum velocity as well.

1) Discrete Particle Swarm Optimization: A discrete multi-valued PSO (DMVPSO) algorithm was proposed by Veeramachaneni *et al.* [19]. In this algorithm, each particle's position is a d-dimensional vector of discrete values in the range $[0, M-1]$ where M is the cardinality of each state variable. The velocity of each particle is a d-dimensional vector of continuous values, as above. A particle's velocity is transformed into a number between $[0, M]$ using the following equation:

$$S_i = \frac{M}{1+\exp(-v_i)}$$

Then each particle's position is updated by generating a random number according to the Gaussian distribution, $x_i \sim N(S_i, \sigma \times (M-1))$ and rounding the result. To ensure the particle's position remains in the range $[0, M-1]$ the following formula is applied:

$$x_i = \begin{cases} M-1 & x_i > M-1 \\ 0 & x_i < 0 \\ x_i & \text{otherwise} \end{cases}$$

III. RELATED WORK

A. Traditional Bayesian Structure Learning

Yuan *et al.* [20] propose an exact algorithm for Bayesian structure learning based on the A* algorithm. This is a state space search technique in which admissible heuristics are used to search only the most promising portions of the search space. The authors propose two heuristics for use with the A* algorithm. The first, involves a relaxation of the acyclicity constraint for Bayesian networks. The second, reduces the relaxation of the first heuristic by preventing directed cycles within some variable groups. While both heuristics are admissible and consistent, the first results in too much relaxation and causes the search space to have a loose bound. While the computational complexity of this algorithm is exponential, the authors' experiments indicate that it outperforms existing exact structure learning methods in terms of both search time and bounding of the search space.

Zheng *et al.* [21] proposed averaged one-dependence estimators (AODE), a Bayesian classification algorithm that addresses the attribute-independence problem of the naive Bayes classifier, which has been shown to be more accurate than traditional naive Bayes in many cases. Unlike other Bayesian classifiers, AODE does not perform model selection, but instead averages the probability of the class node given the attributes over all possible one-dependence Bayesian classifiers to determine the class of each instance.

The Greedy Thick Thinning algorithm (GTT) described by Heckerman [22] is a common approach to Bayesian structure learning. The algorithm begins with a fully connected graph and removes arcs between nodes based on conditional independence tests. GTT then optimizes the structure by modifying the graph and scoring the result. The modifications used by GTT include adding an arc if one does not already exist, and removing or reversing an arc if it already exists. The modified network is then scored, and if the modifications fail to improve the network, the algorithm returns the current structure. To encourage greater exploration, a predetermined number of network perturbations are produced after each scoring.

Heckerman also describes a simulated annealing algorithm for general structure learning [22]. In simulated annealing the system is initialized to some initial temperature T_0. Then some change to the network is randomly chosen and is applied based on the value of $p = \exp(\Delta e/T_0)$, where Δe is the change in fitness. If $\Delta e > 1$, then the change is accepted; otherwise, the change is accepted with probability p. This selection and evaluation process is repeated α times or until we make β changes. If no changes are made in α repetitions, then the search is terminated. Otherwise, the temperature is lowered by multiplying T_0 by a decay factor $0 < \gamma < 1$, and the search process continues. If the temperature has been lowered more than δ times, then the algorithm is terminated.

Cooper *et al.* describe a method for constructing Bayesian networks from data called K2 [14]. K2 is a greedy algorithm that determines the set of edges that best matches the data given a node ordering. If some node X_i precedes node X_j in the ordering, then X_j cannot be a parent of X_i. K2 visits each node X_i based on the sequence specified in the ordering and greedily inserts a parent into the parent set of X_i if the addition of the parent maximizes the score of the network. Their algorithm uses $P(B, D)$ as the scoring function.

B. Multi-population Algorithms

Several authors have proposed multi-population genetic algorithms (GA) [23], [24], [25], [26]. In these models, several subpopulations are maintained by the genetic algorithm, and members of the populations are exchanged through a process called migration. These methods have been shown to obtain better quality solutions than traditional GAs [24] when applied to the problems of neural network parameter learning, the traveling salesman problem, and several deceptive problems proposed by Goldberg *et al.* [27]. Because the islands maintain some independence, each island can explore a different region of the search space while sharing information with other islands through migration. This improves genetic diversity and solution quality [26].

Bergh and Engelbrecht developed several multi-population PSO methods for training multi-layer feedforward neural networks [28]. These methods include NSPLIT in which there is a single particle swarm for each neuron in the network and LSPLIT in which there is a swarm assigned to each layer of the network. The authors' claim that splitting the swarms in this way results in a finer-grained credit assignment, reducing the possibility of neglecting a potentially good solution for a specific component of the solution vector. The results obtained by van den Bergh and Engelbrecht indicate that the distributed algorithms outperform traditional PSO methods. The authors did not compare these methods to any non-PSO approaches. Note, however, that these methods do not include any communication between the swarms and provide access to a global fitness function.

Recently a new distributed approach to improve the performance of the PSO algorithm has been explored where multiple swarms are assigned to overlapping subproblems. This approach is called Overlapping Swarm Intelligence (OSI) [12],

[29], [30]. In OSI each swarm searches for a partial solution to the problem, and solutions found by the different swarms are combined to form a complete solution once convergence has been reached. Where overlap occurs, communication and competition take place to determine the combined solution to the full problem.

Haberman and Sheppard first proposed OSI as a method to develop an energy-efficient routing protocol for sensor networks that ensures reliable path selection while minimizing the energy consumption during message transmission [12]. In this approach a swarm is associated with each node in the sensor network and each swarm consists of a particle for its corresponding node and the particles for all of the nodes immediate neighbors. Thus, the swarms for a given node overlaps with its neighboring swarms. This algorithm was shown to be able to extend the life of the sensor networks and to perform significantly better than current energy-aware routing protocols.

Ganesan Pillai extended the OSI method to learn the weights of deep artificial neural networks [29]. This algorithm separates the structure of the network into paths where each path begins at an input node and ends at an output node. Each of these paths is associated with a swarm that learns the weights for that path of the network. A common vector of weights is maintained across all swarms to describe a global view of the network. This vector is created by combining the weights of the best particles in each of the swarms. This method was shown to outperform the backpropagation algorithm, the traditional PSO algorithm, and both NSPLIT and LSPLIT on deep networks. A distributed version of this approach was developed subsequently by [30].

Fortier and Sheppard developed a method for abductive inference in Bayesian Networks based on OSI [31]. In this approach, multiple swarms are used to find the most probable state assignments for a Bayesian network given the evidence. Each node in the network is associated with a swarm that learns the state assignments for its Markov blanket. Swarms periodically communicate and compete for inclusion in the final set of most probable state assignments.

C. Soft Computing Bayesian Structure Learning

Several researchers have applied soft computing techniques to the problem of Bayesian structure learning. Wang *et al* proposed a PSO algorithm [9] where the position of each particle is an $n \times n$ adjacency matrix with n being the number of variables in the network. The fitness function for a candidate network in this approach is the log likelihood of the data given the model as shown in Equation 2. Before constructing the Bayesian network for fitness evaluation, the adjacency matrix is checked for cycles and the cycles are removed.

An island model genetic algorithm for Bayesian structure learning was proposed by Regnier-Coudert *et al*. In this algorithm, each chromosome consists of an ordering of the nodes in the network [10]. For a given ordering, each node can only be a parent of a node ahead of it in the ordering. A Bayesian network is constructed from an ordering by applying the K2 edge selection algorithm described in [14] to the ordering. The fitness function for a candidate network in this approach is denoted as:

$$F(B) = P(B, D) \tag{5}$$

The populations of chromosomes are separated into several islands running the genetic algorithm in parallel. The search is periodically paused and migration occurs between the islands. In this approach the genetic algorithm acts as a heuristic to learn a variable ordering for the K2 algorithm.

Wu *et al* proposed an ant colony optimization algorithm for Bayesian structure learning called K2ACO [11]. In this algorithm the representation is similar to that used in [10]. Each individual represents a node ordering, and the fitness of each ordering is calculated by running the K2 search algorithm. Each ant in the algorithm traverses a graph of the nodes in the network and an ant in node i moves to node j according to a probabilistic state transition rule based on the amount of pheromone on the edge between i and j. The amount of pheromone deposited on an edge is based on Equation 5. The order of the nodes in the graph traversal containing the highest pheromone levels defines the variable ordering for the K2 algorithm.

IV. OSI Structure Learning

Given a classification problem consisting of a series of attributes A_i and a class label C, a Bayesian classifier is a Bayesian network in which a node is associated with each attribute A_i and the class C. Classification is then performed by computing the most probable state of the class C given the states of the attributes $A_1, ..., A_n$. We propose an algorithm for learning Bayesian network classifiers based on OSI. The pseudocode for our approach is shown in Algorithm 2.

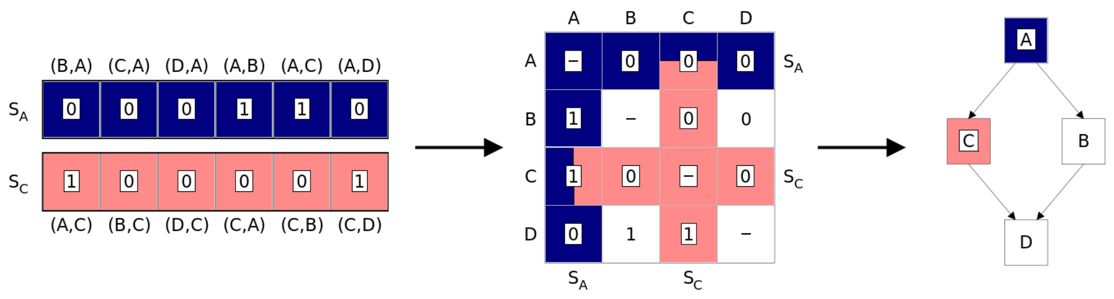
In this algorithm, a swarm is associated with each attribute node in the network. Each node's corresponding swarm learns the input and output edges associated with that node. A global network G is maintained across all swarms for inter-swarm communication. This global network is represented by an $n \times n$ adjacency matrix M. Each element $m_{i,j}$ in the matrix is defined as shown in Equation 6.

$$m_{i,j} = \begin{cases} 1 & \text{if node } j \text{ is a parent of node } i \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The global network is initialized using the Tree Augmented Naive Bayes (TAN) algorithm, and a single particle in each swarm is initialized to the positions defined by the initial global network. This is done to ensure that particles do not initially explore unproductive regions of the search space.

Each particle's position x_i is defined by a d-dimensional vector of binary values where $x_i \in \{0, 1\}^d$, $d = 2n$, and n is the number of nodes in the network. Each position value in the first half of the vector determines if a given variable in the network is a parent of the node, while each position value in the second half of the vector determines if a given variable in the network is a child of the node. For a particle in the swarm associated with node i, the position vector encodes all elements in the column and row associated with node i in the adjacency matrix M. These position values are constrained so that if a node X_i is a parent of X_j then X_i cannot be the child of X_j.

An example illustrating our representation is shown in Figure 1. In the first section of the figure, the representation



(B,A) (C,A) (D,A) (A,B) (A,C) (A,D)
S_A
0 0 0 1 1 0
S_C
1 0 0 0 0 1
(A,C) (B,C) (D,C) (C,A) (C,B) (C,D)
A B C D
S_A
S_C
A
B
C
D

Fig. 1: Representation example

Algorithm 2 OSI

```
1:  Generate $G$ and $M$ using TAN
2:  Initialize particles in each swarm

3:  **repeat**
4:    **for** each swarm $s$ **do**
5:      **for** each particle, $p \in s$ **do**
6:        Add edges of $p$ to $G$
7:        Remove cycles from $G$
8:        Calculate particle fitness $f(p)$
9:        **if** $f(p) >$ $p$'s personal best fitness **then**
10:         Update $p$'s personal best position and fitness
11:       **end if**
12:       **if** $f(p) >$ the global best fitness **then**
13:         Update global best position and fitness for $s$
14:       **end if**
15:       Update $p$'s velocity and position
16:     **end for**
17:   **end for**
18:   **for** each entry $m_{i,j} \in M$ **do**
19:     Let $s_i$ be the swarm associated with node $i$
20:     Let $s_j$ be the swarm associated with node $i$
21:     **if** $s_i$ and $s_j$ conflict over $M_{i,j}$ **then**
22:       $m_{i,j} \leftarrow compete(s_i, s_j)$
23:       **if** $s_i$ wins competition **:** seed $s_i$ with $m_{i,j}$
24:       **else** seed $s_j$ with $m_{i,j}$
25:     **end if**
26:   **end for**
27: **until** termination criterion is met

28: **return** $A$
```

used by the particles in swarms S_A and S_C is shown, and the edge represented by each position value is indicated. Our algorithm inserts the position vectors for the particles in swarms S_A and S_C into an adjacency matrix, as shown in the second section of the figure. In the adjacency matrix, positions learned by swarm S_C are shown in the lighter color, while positions learned by swarm S_A are shown in the darker color. Entries in the matrix that are learned by both swarms are highlighted with both colors. Once the particle positions have been inserted into the adjacency matrix, a network containing the edges defined by the original particle positions can be constructed as shown in the third section of the figure.

To insert a substructure S_i for a node X_i into the network, all edges connecting to X_i are removed and the edges encoded in S_i are added to the network. The is done by inserting the row and column elements defined by a particle's position vector into the global adjacency matrix M. From this adjacency matrix a Bayesian network is constructed and an outgoing edge is inserted from each attribute A_i and the class C.

When inserting the substructure of a particle into the network, cycles may be introduced. To prevent this, a two step process is performed each time a substructure is inserted into the global network. First, we locate the cycles using the following process:

- Identify a source node (a node that either has no incoming edges or no outgoing edges).
- Delete the source node and its edges.
- Repeat until there are no source nodes.

Once this process is complete the remaining network consists of the existing cycles. Next each cycle is broken by randomly selecting one of the nodes in the cycle and removing one of its remaining outgoing edges.

For this problem, two swarms are said to overlap if their highest scoring particle positions conflict. The positions of two particles p_i and p_j are said to conflict if, for a given entry $m_{i,j}$ in the adjacency matrix M, p_i encodes a different value for $m_{i,j}$ than p_j. At the end of each iteration of the algorithm, a competition is held between overlapping swarms to determine which set of edges is inserted into the network. This competition is held between the conflicting position values of the most fit particles of each competing swarm. The values resulting in the best fitness score are included in the global adjacency matrix M.

After each competition, the swarm resulting in the lowest fitness is seeded with the position of the winning swarm. This involves replacing the position value for the lowest ranked invidual in the swarm with the value in the global solution.

A. *Fitness Evaluation*

We compared four fitness functions to evaluate the quality of a network. The first fitness function (OSI-LL) is the log likelihood of the data given the model as defined in Equation 2. The second fitness function (OSI-BIC) is Bayesian information criterion described in Equation 3. The third fitness function

(OSI-ERR) is the classification error of the network when used to classify the training data as defined in Equation 7.

$$error = \frac{\sum_{i=1}^{|C|} F_i}{\sum_{i=1}^{|C|} F_i + T_i} \tag{7}$$

where F_i is the number of examples incorrectly classified as class i and T_i is the number of examples correctly classified as class i. The final fitness function (OSI-CMI) is the average conditional mutual information of each edge in the network given the class as defined by Equation 8.

$$ACMI = \frac{\sum_{(X,Y)\in E} I(X,Y|C)}{|E|} \tag{8}$$

where E denotes the set of edges in the network, (X,Y) denotes the edge between nodes X and Y, and $I(X,Y|C)$ is the conditional mutual information between nodes X and Y given the class C.

The fitness of particle p is defined as the quality of the network G when the substructure defined by the particle's parameters is inserted into the network.

B. Computational Complexity

We first derive the computational complexity of OSI and show which step has the most computational burden in the algorithm. To do so, we break down each step in the algorithm. We refer to the pseudocode shown in Algorithm 2.

- **Cycle Removal—**Cycle removal (line 7) can be performed in $\mathcal{O}(|V|^2)$ where V is the set of verticies. Let n be the number of entries in the adjacency matrix. Then $|V| = \sqrt{n}$, and the complexity of cycle removal is $\mathcal{O}(n)$.
- **Fitness Evaluation—**Next, we determine the complexity of evaluating the fitness of an individual particle (line 8). To do so, we approximate the fitness function complexity as $\mathcal{O}(n\Lambda)$, where n is number of entries in the adjacency matrix, and Λ denotes the computational complexity specific to the fitness function used to evaluate each particle. When this is combined with cycle removal the time complexity becomes $\mathcal{O}(n\Lambda + n) = \mathcal{O}(n\Lambda)$
- **Optimization Algorithm—**Next, we determine the complexity of the underlying PSO algorithm (lines 5–16). We assume that the algorithm has $m = |P|$ individuals, where P is the set of all the individuals in the population. During each iteration, an individual with n variables has its position and fitness updated. These two steps are done sequentially, with updating the position having a complexity of $\mathcal{O}(n)$ and while evaluating the fitness is $\mathcal{O}(n\Lambda)$. This is done m times, once for for each individual; therefore, assuming the algorithm performs L iterations, the total complexity is $\mathcal{O}(n\Lambda Lm)$.

To complete our analysis of the complexity of OSI, we next show the complexity of the two main parts of OSI: solve and competition.

- **Solve—**The solve step in OSI (lines 4–17) involves iterating over the set of subpopulations $\boldsymbol{S}$ and having each one optimize over its variables. Using the complexity from above for optimization algorithms, each subpopulation has a complexity of $\mathcal{O}(nLm)$. Since this is done $s = |\boldsymbol{S}|$ times, the total complexity of this step is $\mathcal{O}(sn\Lambda Lm)$.
- **Competition—**In OSI, the competition step (lines 18–24) is used to find the optimal set of values for the variables in X. This is done by iterating over all the variables and then comparing the fitness of the two competing individuals. Note that there are n variables and 2 subpopulations while each fitness evaluation has a complexity of $\mathcal{O}(n\Lambda)$. Therefore, the total complexity of the competition step in OSI is $\mathcal{O}(2n^2\Lambda) = \mathcal{O}(n^2\Lambda)$.

Thus OSI iterates over the solve and competition steps k times. Combining the steps above and multiplying times k gives a complexity of

$$\begin{aligned}\mathcal{O}(\text{OSI}) &= \mathcal{O}(k(n\Lambda Lms + n^2\Lambda))\\ &= \mathcal{O}(kn\Lambda Lms + kn^2\Lambda)\end{aligned}$$

In the algorithm presented above, a swarm is associated with each attribute in the classification problem, therefore $s = n$. Substituting this into the above equation, we now have

$$\begin{aligned}\mathcal{O}(\text{OSI}) &= \mathcal{O}(kn^2\Lambda Lm + kn^2\Lambda)\\ &= \mathcal{O}k(n^2\Lambda Lm)\end{aligned}$$

Based on these results, we can see that OSI will almost always be more computationally complex due to the n^2 factor. However, the parameters k, L, and m in OSI can be set such that it becomes competitive with PSO.

V. Experiments

A. Methodology

To evaluate our algorithm, several experiments were performed using datasets from the UCI Machine Learning Repository [32]. These experiments focused on comparing our algorithm with the PSO algorithm discussed in [9], the Tree Augmented Naive Bayes algorithm [17], and the Greedy Thick Thinning algorithm [22]. Our comparisons also include modification of the PSO algorithm discussed in [9] using each of the fitness functions described above. We compared these algorithms to four different versions of our own algorithm using each fitness function described in Section 4.1.

For the OSI algorithms, five particles were assigned to each sub-swarm. For the single swarm PSO, the total number of particles was five times the number of features, to ensure a fair comparison between the algorithms. For both swarm-based algorithms, ϕ_1 and ϕ_2 were set to 1.49618, while ω was set to 0.7298. Eberhart and Shi empirically determined that these are good parameter choices for ω, ϕ_1, and ϕ_2 [33].

To evaluate the performance of the algorithms, the data was divided into training and testing data sets using a 5×2 cross-validation procedure. We performed pairwise comparisons of average classification accuracy using a paired Student t-test for each pair of algorithms. For all t-tests we used a 95% confidence interval. We hypothesize that OSI will outperform

TABLE I: Datasets used in experiments

Dataset	Attributes	Classes	Instances
Ecoli	5	8	334
Vote	16	2	435
Nursery	8	5	12960
Car	6	4	1728
CMC	9	3	1473
TicTacToe	9	2	958
Spect	22	2	267

traditional PSO for most datasets and, when the proper fitness function is chosen, OSI will outperform both TAN and Greedy.

These experiments were performed using seven datasets: Nursery, Car, CMC, TicTacToe, Spect, Vote, and Ecoli. We selected datasets with large numbers of instances, and most selected datasets have few or no continuous attributes. For the datasets containing continuous attributes, each continuous attribute was discretized into five bins using equal frequency binning. The information about the datasets used is summarized in Table I.

B. Results

Tables II and III show the average classification accuracy for each algorithm and each dataset. Bold values indicate that the corresponding algorithm's performance is statistically significantly better than all other algorithms for the dataset. Algorithms that tie statistically for best are bolded.

Table II compares the classification accuracy of the swarm based approaches. For all datasets, either OSI-ERR or OSI-CMI tie statistically for best. Also, for all but two of the datasets, the OSI algorithm with the highest classification accuracy outperforms traditional PSO using the same fitness function.

Table III compares the average classification accuracy of TAN, GREEDY, and the swarm based algorithms that most frequently achieved the highest classification accuracy. For the Vote dataset, all algorithms other than PSO-ERR tie statistically for best. For the TicTacToe dataset OSI-ERR ties statistically with PSO-ERR for best. OSI-CMI outperforms all competing algorithm for every other dataset.

C. Discussion

The paired t-tests on the classification accuracy indicate that either OSI-ERR, or OSI-CMI performed better than or equivalent to the other methods for every dataset. These results show that OSI-CMI outperforms the other methods for three of the datasets. OSI-ERR and OSI-CMI are the methods that most frequently outperformed the other approaches and these algorithms outperformed TAN, Greedy, and the competing PSO algorithms on all but two of the datasets. This indicates that both OSI-ERR and OSI-CMI have an advantage when used to learn the structures of Bayesian classifiers. More importantly, an OSI-based method was always listed as one of the winning algorithms.

We also note that our paired t-tests showed that the OSI algorithms outperformed the single swarm PSO algorithms for many of the datasets, while single swarm PSO algorithms never significantly outperformed the OSI algorithm using the same fitness function. These results indicate that the communication and competition introduced by OSI gives the algorithm an advantage over traditional PSO.

Since multiple swarms learn the structure for a single node, our approach ensures greater exploration of the search space, which results in improved performance in terms of classification accuracy. Also, since the swarms are split over the nodes of the network, the possibility of neglecting a potentially good sub-structure for a specific component of a complete structure is reduced.

The results in Table II also indicate that no fitness function clearly outperforms the others for all data sets. However, conditional mutual information appears to allow OSI to have the best performance for five of the seven datasets. In all cases, when OSI-CMI failed to achieve the best score, OSI-ERR had the best performance. These results indicate that, when average conditional mutual information fails to provide good performance, classification accuracy may be a good alternative scoring metric.

VI. Conclusions and Future Work

We have presented a swarm-based algorithm for the learning of Bayesian network classifiers. In this algorithm a swarm is associated with each attribute in the data, and that swarm learns parent/child edges for its attribute node. We compared our algorithms to several other approaches to Bayesian network structure learning. Our results indicate that OSI significantly outperforms the other approaches on several of the datasets studied in terms of classification accuracy and that at least one OSI-based method is always among the winners.

For future work we plan to extend our approach to the problem of general structure learning. We will compare our more general approach to structure learning with several other structure learning algorithms such as the K2 algorithm [14] and its variations [14], [11]. We will also apply OSI to the problems of parameter estimation and latent variable learning. We are also developing the theory of the general OSI. We plan to study the convergence of OSI by drawing from the approaches of [34] and [35] on the distributed consensus problem to analyze the relationship between OSI convergence rate and the structure of the sub-swarms. We will also empirically verify the existence of optimal swarm overlap structures and are developing methods to identify good overlap structures for OSI when applied to a given problem.

References

[1] M. Zou and S. D. Conzen, "A new dynamic bayesian network (dbn) approach for identifying gene regulatory networks from time course microarray data," *Bioinformatics*, vol. 21, no. 1, pp. 71–79, 2005.

[2] G. Agre, "Diagnostic bayesian networks," *Computers and Artificial Intelligence*, vol. 16, no. 1, 1997.

[3] U. Lerner, R. Parr, D. Koller, G. Biswas *et al.*, "Bayesian fault detection and diagnosis in dynamic systems," in *AAAI/IAAI*, 2000, pp. 531–537.

[4] G. D. Kleiter, "Bayesian diagnosis in expert systems," *Artificial Intelligence*, vol. 54, no. 1, pp. 1–32, 1992.

[5] D. Nikovski, "Constructing bayesian networks for medical diagnosis from incomplete and partially correct statistics," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 4, pp. 509–516, 2000.

TABLE II: Comparison of classification accuracy for Swarm algorithms

Dataset	OSI-LL	PSO-LL	OSI-CMI	PSO-CMI	OSI-BIC	PSO-BIC	OSI-ERR	PSO-ERR
Vote	0.908	0.908	**0.929**	**0.925**	**0.921**	**0.920**	**0.916**	0.907
TicTacToe	**0.882**	0.856	0.737	0.721	0.727	0.745	**0.896**	**0.881**
Nursery	**0.952**	0.940	0.910	0.914	0.911	0.916	**0.958**	0.945
Car	0.860	0.849	**0.891**	0.870	0.837	0.849	0.823	0.788
CMC	0.473	0.464	**0.517**	0.479	0.510	0.511	0.473	0.470
Spect	0.758	0.776	**0.815**	0.796	0.793	0.793	0.779	0.763
Ecoli	0.734	0.730	**0.809**	0.795	**0.813**	0.799	0.713	0.688

TABLE III: Comparison of classification accuracy against TAN and Greedy

Dataset	OSI-CMI	PSO-CMI	OSI-ERR	PSO-ERR	TAN	Greedy
Vote	**0.929**	**0.925**	**0.916**	0.907	**0.934**	**0.935**
TicTacToe	0.737	0.721	**0.896**	**0.881**	0.746	0.755
Nursery	0.910	0.914	**0.958**	0.945	0.909	0.896
Car	**0.891**	0.870	0.860	0.849	0.871	0.867
CMC	**0.517**	0.479	0.473	0.464	0.506	0.448
Spect	**0.815**	0.796	0.758	0.776	0.801	0.750
Ecoli	**0.809**	0.795	0.734	0.730	0.760	0.791

[6] I. Kononenko, "Inductive and bayesian learning in medical diagnosis," *Applied Artificial Intelligence*, vol. 7, no. 4, pp. 317–337, 1993.

[7] P. D. Donnelly and J. W. Sheppard, "Classification of musical timbre using bayesian networks," *Computer Music Journal*, vol. 37, no. 4, pp. 70–86, 2013.

[8] D. M. Chickering, "Learning bayesian networks is np-complete," in *Learning from data*. Springer, 1996, pp. 121–130.

[9] T. Wang and J. Yang, "A heuristic method for learning bayesian networks using discrete particle swarm optimization," *Knowledge and information systems*, vol. 24, no. 2, pp. 269–281, 2010.

[10] O. Regnier-Coudert and J. McCall, "An island model genetic algorithm for bayesian network structure learning," in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2012, pp. 1–8.

[11] Y. Wu, J. McCall, and D. Corne, "Two novel ant colony optimization approaches for bayesian network structure learning," in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2010, pp. 1–7.

[12] B. K. Haberman and J. W. Sheppard, "Overlapping particle swarms for energy-efficient routing in sensor networks," *Wireless Networks*, vol. 18, no. 4, pp. 351–363, 2012.

[13] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[14] G. F. Cooper and E. Herskovits, "A bayesian method for the induction of probabilistic networks from data," *Machine learning*, vol. 9, no. 4, pp. 309–347, 1992.

[15] G. Schwarz *et al.*, "Estimating the dimension of a model," *The annals of statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[16] E. Wit, E. v. d. Heuvel, and J.-W. Romeijn, "all models are wrong...: an introduction to model uncertainty," *Statistica Neerlandica*, vol. 66, no. 3, pp. 217–236, 2012.

[17] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[18] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. IEEE, 1995, pp. 39–43.

[19] K. Veeramachaneni, L. Osadciw, and G. Kamath, "Probabilistically driven particle swarms for optimization of multi-valued discrete problems: Design and analysis," in *Proceedings of the IEEE Swarm Intelligence Symposium*, 2007, pp. 141–149.

[20] C. Yuan and B. Malone, "Learning optimal bayesian networks: A shortest path perspective," *Journal of Artificial Intelligence Research*, vol. 48, pp. 23–65, 2013.

[21] F. Zheng and G. I. Webb, "Averaged one-dependence estimators," *Encyclopedia of Machine Learning*, pp. 63–64, 2010.

[22] D. Heckerman, "A tutorial on learning with bayesian networks," in *Innovations in Bayesian Networks*. Springer, 2008, pp. 33–82.

[23] R. Tanese, J. Co-Chairman-Holland, and Q. Co-Chairman-Stout, "Distributed genetic algorithms for function optimization," in *Proceedings of the International Conference on Genetic Algorithms*. University of Michigan, 1989, pp. 434–439.

[24] D. Whitley and T. Starkweather, "Genitor II: A distributed genetic algorithm," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 2, no. 3, pp. 189–214, 1990.

[25] T. Belding, "The distributed genetic algorithm revisited," in *Proceedings of the International Conference on Genetic Algorithms*, 1995, pp. 114–121.

[26] D. Whitley, S. Rana, and R. Heckendorn, "The island model genetic algorithm: On separability, population size and convergence," *Journal of Computing and Information Technology*, vol. 7, pp. 33–48, 1999.

[27] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.

[28] F. van den Bergh and A. Engelbrecht, "Cooperative learning in neural networks using particle swarm optimizers," *South African Computer Journal*, vol. 26, pp. 94–90, 2000.

[29] K. G. Pillai and J. W. Sheppard, "Overlapping swarm intelligence for training artificial neural networks," in *Proceedings of the IEEE Swarm Intelligence Symposium*, April 2011, pp. 1–8.

[30] N. Fortier, J. W. Sheppard, and K. G. Pillai, "DOSI: Training artificial neural networks using overlapping swarm intelligence with local credit assignment," in *Joint 6th International Conference on Soft Computing and Intelligent Systems (SCIS) and 13th International Symposium on Advanced Intelligent Systems (ISIS)*. IEEE, 2012, pp. 1420–1425.

[31] N. Fortier, J. Sheppard, and S. Strasser, "Abductive inference in bayesian networks using distributed overlapping swarm intelligence," *Soft Computing*, pp. 1–21, 2014.

[32] D. N. A. Asuncion, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html

[33] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, vol. 1. IEEE, 2000, pp. 84–88.

[34] S. Patterson, B. Bamieh, and A. El Abbadi, "Convergence rates of distributed average consensus with stochastic link failures," *IEEE Transactions on Automatic Control*, vol. 55, no. 4, pp. 880–892, 2010.

[35] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.