Parameter Estimation in Bayesian Networks Using Overlapping Swarm Intelligence

Nathan Fortier, John Sheppard, and Shane Strasser
Montana State University
Bozeman MT, 59717-3880
nathan.fortier@msu.montana.edu, john.sheppard@cs.montana.edu,
shane.strasser@cs.montana.edu

ABSTRACT

Bayesian networks are probabilistic graphical models that have proven to be able to handle uncertainty in many real-world applications. One key issue in learning Bayesian networks is parameter estimation, i.e., learning the local conditional distributions of each variable in the model. While parameter estimation can be performed efficiently when complete training data is available (i.e., when all variables have been observed), learning the local distributions becomes difficult when latent (hidden) variables are introduced. While Expectation Maximization (EM) is commonly used to perform parameter estimation in the context of latent variables, EM is a local optimization method that often converges to sub-optimal estimates. Although several authors have improved upon traditional EM, few have applied population based search techniques to parameter estimation, and most existing population-based approaches fail to exploit the conditional independence properties of the networks. We introduce two new methods for parameter estimation in Bayesian networks based on particle swarm optimization (PSO). The first is a single swarm PSO, while the second is a multi-swarm PSO algorithm. In the multi-swarm version, a swarm is assigned to the Markov blanket of each variable to be estimated, and competition is held between overlapping swarms. Results of comparing these new methods to several existing approaches indicate that the multi-swarm algorithm outperforms the competing approaches when compared using data generated from a variety of Bayesian networks.

Categories and Subject Descriptors

Swarm Intelligence [**Probabilistic Graphical Models**]: Bayesian Networks

1. INTRODUCTION

Bayesian networks are widely used models for reasoning under uncertainty that provide a compact representation of high-dimensional joint probability distributions. There are two categories of learning related to Bayesian networks, the first involves learning the structure of the network, while the second involves learning the parameters that define the local probability distributions of the network. In the case of complete data, parameter estimation is not difficult, but in practice training data is often incomplete and some hidden variables never have data.

It is common for latent or hidden variables to be incorporated into Bayesian network models. Latent variables allow the network to encode unobserved effects and can drastically reduce the number of parameters used to specify the model. However, when latent variables are introduced to a Bayesian network, marginal probabilities can no longer be computed efficiently [14]. Also, since the likelihood of the parameters is no longer decomposable, conditional dependencies exist between the optimal parameters for the network's variables.

While several algorithms have been proposed for parameter estimation, these algorithms have a tendency to gravitate towards sub-optimal solutions, resulting in poor generalization for unseen test data, especially when estimating parameters for joint distributions with many conditional dependencies. Also, despite their effectiveness on complex search problems, few authors have applied population based search methods to the problem of parameter estimation.

We propose a new algorithm for latent variable parameter estimation in Bayesian networks based on particle swarm optimization (PSO). Our algorithm assigns a single swarm to each latent variable and its children. Each swarm learns the unknown parameters for the corresponding node's Markov blanket. Swarms with overlapping Markov blankets compete for inclusion in a global parameter set.

We hypothesize that, by assigning a swarm to the Markov blanket of a node, we can exploit conditional independence properties of the network to achieve better performance than single population search methods. To evaluate this hypothesis, we give a traditional single-swarm approach to parameter estimation and compare our multi-swarm approach to single-swarm PSO and several other existing approaches to parameter estimation. We also hypothesize that both of our PSO-based methods will yield results competitive with traditional and state-of-the-art methods based on methods that extend the Expectation-Maximization (EM) algorithm.

2. BACKGROUND

2.1 Bayesian Networks

A Bayesian network is a directed acyclic graph that represents a joint distribution over a set of variables [14]. In a

GECCO '15, July 11 - 15, 2015, Madrid, Spain

 ISBN 978-1-4503-3472-3/15/07…$15.00
DOI: http://dx.doi.org/10.1145/2739480.2754793

Bayesian network, each random variable is represented by a node, and edges between nodes represent conditional dependence relationships between the variables. Each root node encodes a prior probability distribution, while each non-root node encodes a probability distribution conditioned on the node's parents. In discrete Bayesian networks, these distributions are represented as conditional probability tables (CPT). For the variables in the network, the probability of any entry in 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 network, which exploit conditional independence properties of the variables, the joint distribution can be represented equivalently as

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

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

In a Bayesian network, the Markov blanket of a node consists of the node's parents, children, and children's parents. A variable X_i is conditionally independent of all other variables in the network given its Markov blanket.

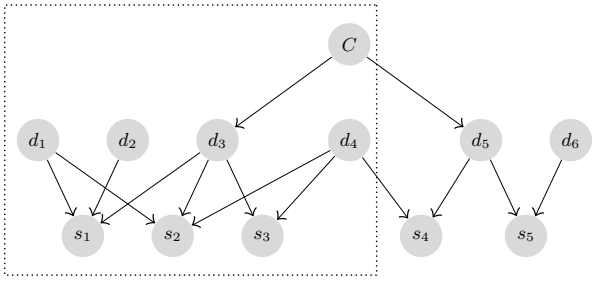
$$\{X_i \perp (\mathbf{X} \backslash (\{X_i\} \cup MB(X_i))) \, | MB(X_i)\}$$

An example illustrating the concept of a Markov blanket is shown in Figure 1. Figure 1(a) shows the Markov blanket of d_3, Figure 1(b) shows the Markov blanket of d_5, and Figure 1(c) shows the Markov blankets of both d_3 and d_5. In the example, nodes in the Markov blanket of d_3 and nodes in the Markov blanket d_5 are shown with a dashed rectangle. In Figure 1(c) nodes that are in the Markov blankets of both d_3 and d_5 (namely c and d_4) are shown to be inside both dashed rectangles, thus indicating an overlap. We will exploit these overlaps later.

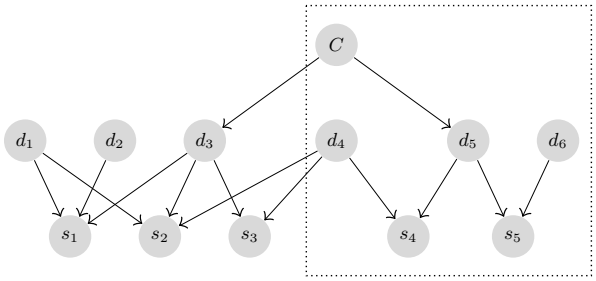
2.2 Particle Swarm Optimization

Particle Swarm Optimization is a population based search technique proposed by Kennedy and Eberhart [4] in 1995, which is inspired by the behavior of bird flocks and fish schools. The pseudocode for PSO is presented in Algorithm 1. In PSO, the population is initialized with a number of random solutions called particles. Each particle contains a position vector that encodes a potential solution in the search space and a velocity vector that defines how the particles will move through the search space. Each particle tracks the coordinates in the search space 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 found so far by any particle in the swarm. This is called the global best position and is denoted p_g.

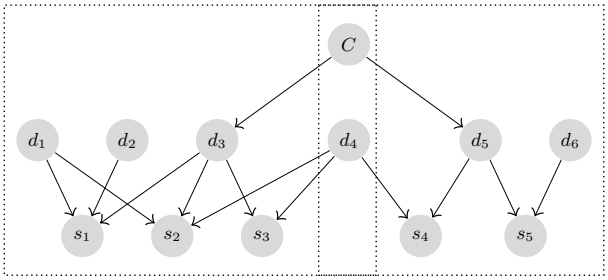
Typically, both position and velocity are defined as vectors of real numbers. The search process updates the position vector of each particle based on that particle's corresponding velocity vector. These velocity vectors are updated at each iteration based on the fitness of the states visited by the particles. Eventually all particles should move closer to a local optimum in the search space.



(a) Markov blanket of d_3



(b) Markov blanket of d_5



(c) Overlap of Markov blankets for d_3 and d_5

Figure 1: Markov blanket example

3. RELATED WORK

There have been prior methods applying swarm and evolutionary based methods to Bayesian networks. These methods include work like learning Bayesian network structures using Ant Colony Optimization [17] and abductive inference using PSO [8]. However, to our knowledge, no work has been published using PSO or multi-population methods for parameter estimation. In this next section, we discuss work related to parameter estimation and multi-population optimization algorithms

3.1 Parameter Estimation

A number of algorithms have been developed for parameter estimation. The most common algorithm for parameter estimation is Expectation Maximization (EM) [14]. The algorithm begins with an initially random parameter assignment and repeatedly executes the expectation and maximization steps. During the expectation step, the algorithm fills in the missing values using the current parameter estimates of the model. This way, a set of data can be generated with values corresponding, not only to the observable vari-

Algorithm 1 Particle Swarm Optimization

```
repeat
  for each particle position x_i ∈ 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 = ωv_i + U(0, φ_1) ⊗ (p_i − x_i) + U(0, φ_2) ⊗ (p_g − x_i)
    x_i = x_i + v_i
  end for
until termination criterion is met
```

ables but to the hidden variables as well. The maximization step uses the resulting completed data set to find a new maximum likelihood estimate of the probability estimates for those hidden variables. Unfortunately, since the EM algorithm is a local search method, this process often converges to sub-optimal parameters and even small changes to the initial parameters can change the local optima found by the algorithm significantly. Also, the expectation step can often be computationally expensive since it must estimate the joint distribution for each data point.

Several authors have developed enhanced versions of EM in terms of both computational complexity and quality of learned parameters. In 1990 Wei and Tanner proposed a randomized EM algorithm in which the expectation step is approximated using Monte Carlo sampling [21]. In this approach, the data is completed by sampling from the conditional distribution of the missing data for each data point. The expectation is then approximated as the Monte Carlo average. The performance of Monte Carlo EM (MCEM) is often comparable to that of traditional EM and the algorithm has been shown to perform well even when a single sample is drawn for each data-point[2].

In 2002, Elidan *et al.* used data perturbation to improve upon the quality of the local maxima reached by EM [7]. Their algorithm allows EM to escape local maxima by perturbing the training data, thereby forcing the algorithm to explore new ascent directions. This work evaluates the effectiveness of both random data perturbation and adversarial data perturbation, in which the data is modified to directly challenge the current parameter estimates.

In 2005, Elidan and Friedman proposed the information bottleneck EM algorithm for learning both the parameters and the structure of Bayesian networks in the presence of incomplete data [6]. This approach is based on the information bottleneck framework and involves grouping observed variables by mutual information and then creating a hidden variable for each group.

More recently, EM has been combined with population based approaches to improve the quality of learned parameters. In 2006, Jank proposed a genetic algorithm version of EM (GAEM) based on MCEM. In this algorithm, each individual in the population encodes a set of parameter estimates. The fitness of an individual is the approximate probability of the data given the parameters as computed by a single iteration of MCEM[13].

Another variant of EM based on evolutionary computation was proposed by Mengshoel *et al.* in 2012 [15]. This algorithm is described as an age-layered EM (ALEM) approach that discards low likelihood runs before convergence. This algorithm maintains a population of individuals that represent EM runs. The population is then divided into a set of layers, each with a user-defined age limit. Once the number of iterations for an individual EM run exceeds this limit, the individual is removed from the layer and, if the likelihood of the data given the parameters is high enough, the individual is moved to the next layer.

3.2 Multi-population Algorithms

Several authors have proposed multi-population genetic algorithms (GA) [1, 19, 22, 23]. These include island models, in which 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 [23] when applied to the problems of neural network parameter learning, the traveling salesman problem, and several deceptive problems proposed by Goldberg *et al.* [11]. Because the sub-populations 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 [22].

Van den Bergh and Engelbrecht developed several multi-population PSO methods for training multi-layer feed-forward neural networks [20]. 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 these algorithms outperform traditional PSO methods specifically on neural network training problems.

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) [10, 12, 16]. 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 node's immediate neighbors. The particles encode the cost estimate of sending a messing along one of the outgoing edges of the node. Using this method, the swarm 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 and Sheppard extended the OSI method to learn the weights of deep artificial neural networks [16].

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, a traditional single-swarm PSO algorithm, and both NSPLIT and LSPLIT on deep networks. A distributed version of this approach was developed subsequently by [10].

Following the success of OSI on training deep neural networks, Fortier *et al.* adapted the OSI to yield a method for both partial and full abductive inference in Bayesian Networks [8]. 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.

In 2014, a method for learning the structure of Bayesian classifiers was proposed by Fortier *et al.* [9]. In this work, a swarm is assigned to each node in the network, and the swarm learns the parent and child edges for that node. Swarms then compete for inclusion in a global network structure.

4. APPROACH

We describe two algorithms for parameter estimation in Bayesian networks using PSO. The first uses a traditional single swarm approach to solve the problem, while the second uses a multi-population variant of PSO based on OSI.

4.1 Traditional Particle Swarm Optimization

Since PSO has not been applied to parameter estimation in Bayesian networks, we have developed both a single swarm and and OSI-based approach to the problem, thus allows us to compare these two methods in terms of the quality of learned parameters. For the single swarm approach, each particle's position vector x_i is a d-dimensional vector of real numbers where $x_i \in [0,1]^d$, d is the number of unknown parameters in the network, and initially each component of x_i is drawn from a uniform probability distribution $U(0,1)$. Each value corresponds to a parameter in the conditional distribution of a missing variable or the child of a missing variable and the size if this vector is equal to the number of parameters to be estimated.

For some variable A in the network, let $x_{a|\mathrm{Pa}(A)}$ be the value in a particle's position vector that encodes the likelihood of $A = a$ given some state assignment to the parents of A, $\mathrm{Pa}(A)$. During fitness evaluation, $x_{a|\mathrm{Pa}(A)}$ is normalized to compute the particle's estimate of the probability $P(a|\mathrm{Pa}(A))$ as follows:

$$P(a|\mathrm{Pa}(A)) = \frac{x_{a|\mathrm{Pa}(A)}}{\sum_{a' \in A} x_{a'|\mathrm{Pa}(A)}}$$

The parameters for each variable being learned can be computed from the particle's position in this way.

If a variable is present in the data and none of its parents are latent variables, we estimate its parameters directly based on frequency. Given some data-set D we can estimate any joint probability over the observed variables as

$$P(x_1, x_2, ..., x_n) = \frac{\sum_{d \in D} \mathbf{1}_d(x_1, x_2, ..., x_n) + \alpha}{|D| + \alpha z}$$

where

$$z = \prod_i^n |Val(X_i)|$$

In the above equations, $\mathbf{1}_d$ is an indicator function for a data point d, $Val(X)$ is the set of possible values for variable X, and $\alpha > 0$ is a smoothing parameter. Using the above joint probability estimate we can compute any conditional probability over a subset of the observed variables as follows:

$$P(x|\mathrm{Pa}(x)) = \frac{P(x, \mathrm{Pa}(x))}{P(\mathrm{Pa}(x))}$$

Because these probabilities can be computed directly, PSO will not optimize the parameters for observed variables with no latent parents.

Once the parameters have been computed, each particle's fitness is evaluated. To compute the fitness of a set of parameters we use variable elimination to calculate the log likelihood of the data D given the parameters Θ.

$$L(D|\Theta) = \log P(D|\Theta). \quad (1)$$

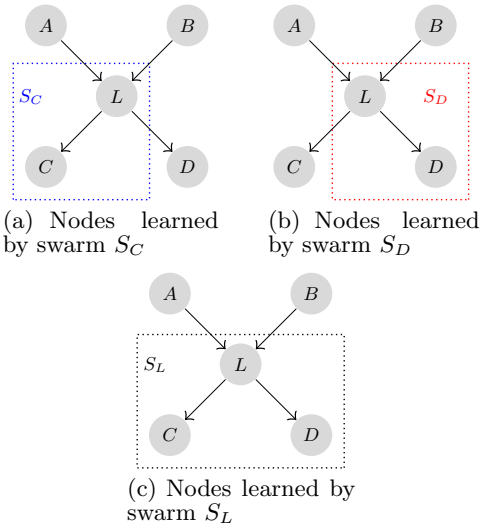
Since performing variable elimination can be computationally expensive, it may be necessary to sample from the training data to obtain a smaller dataset S. The log likelihood of this reduced data set can then be used as the fitness function. Alternatively, an approximate inference method such as importance sampling can also be used. After computing the fitness for a particle, its position and velocity are updated as shown in Algorithm 1.

4.2 Overlapping Swarm Intelligence

We also developed a multi-swarm approach to parameter estimation based on OSI. In this approach, a swarm is assigned to each latent variable, and each child of a latent variable. Each variable's corresponding swarm learns the parameters associated with that variable's Markov blanket using PSO. This representation is advantageous since every node in the network is conditionally independent of all other nodes when conditioned on its Markov blanket.

Figure 2 shows an example network with a single hidden variable L where the nodes whose parameters are learned by a particular swarm are indicated by a dashed rectangle. For this network, the algorithm maintains three swarms: one for the hidden variable L, and one for each of its children, C and D. Figure 2(a) shows that parameters will be learned by swarm S_C, which covers nodes L and C; 2(b) shows that parameters will be learned by swarm S_D, which covers nodes L and D; and Figure 2(c) shows that parameters will be learned by swarm S_L, which in this case covers the nodes in $S_C \cup S_D$. If C and D had children, then S_C and S_D would have included those children while S_L would not. Note that the parameters associated with nodes A and B are not learned by any of the swarms, as these parameters can be estimated directly using the frequency based approach described in the previous section.

The pseudocode for our approach is shown in Algorithm 2. The algorithm is split into two main loops that are executed repeatedly until some termination criterion is met.



(a) Nodes learned by swarm S_C

(b) Nodes learned by swarm S_D

(c) Nodes learned by swarm S_L

Figure 2: Swarm assignment example

The first iterates over each of the swarms, evaluating the fitness of the corresponding particles, while the second holds a competition between the best particles in the swarms to update a global set of network parameters. This set of initially random global parameters is used for inter-swarm communication.

Each particle's position x_i is defined by a d-dimensional vector of continuous values where $x_i \in [0,1]^d$, where d is the number of parameters to be learned by the particle's swarm. Each position value encodes a likelihood value for the probability distribution of a node in the swarm's Markov blanket, thus each particle encodes the parameter estimates for part of the network. As in the traditional PSO algorithm, the log-likelihood of the data given the parameters is used to evaluate the fitness of each particle.

For some swarm s, let $M(s)$ be the set of variables in the Markov blanket associated with s. Let $\Theta_g = \{\theta_{g,v_1}, \ldots, \theta_{g,v_n}\}$ be the set of CPTs for all variables $v_1, \ldots, v_n$ in the network. Let $\Theta_p = \{\theta_{p,v} | v \in M(s)\}$ be the set of CPTs encoded by the position vector of particle p in swarm s. A new full parameter set $\Theta_{g,p}$ can be constructed by inserting Θ_p into Θ_g as follows:

$$\Theta_{g,p} = \Theta_p \cup \Theta_g \backslash \{\theta_{g,v} | v \in M(s)\}$$

We use the log likelihood of the data given the parameters $\Theta_{g,p}$ as the fitness for the particle p.

$$f(p) = L(D|\Theta_{g,p})$$

This function defines the fitness of particle p as the log likelihoods of the data given the global parameters Θ_g, when the parameter estimates encoded in p are substituted into Θ_g.

When multiple swarms learn the parameters for a given node, (such as c and d_4 in Figure 1) these swarms are said to overlap. After each iteration of the algorithm, overlapping swarms compete to determine which parameter estimates to include for each entry in the CPTs. This competition is held between the parameter estimates encoded in the personal best particles of each swarm. The parameters resulting in the highest log-likelihood are selected for inclusion in the global parameter set. Thus in the example presented in Figure 1, the swarms associated with d_3 and d_5 would compete to determine which parameter values are assigned to the nodes c and d_4.

Algorithm 2 Overlapping Swarm Intelligence

```
Randomly initialize Θ_g
Initialize particles in each swarm

repeat
  for each swarm s do
    for each particle, p ∈ s do
      Construct Θ_{g,p}
      Calculate particle fitness f(p)
      if f(p) > p's personal best fitness then
        Update p's personal best position and fitness
      end if
      if f(p) > the global best fitness then
        Update global best position and fitness for s
      end if
      Update p's velocity and position
    end for
  end for

  for each incomplete variable v in the network do
    Let the set of swarms S = {s|v ∈ MB(s)}
    for each parameter z ∈ θ_{g,x} do
      z ← Compete(S, z)
    end for
  end for
until termination criterion is met

return Θ_g
```

Algorithm 3 $Compete(S, z)$

```
L_best ← −∞
for each swarm s ∈ S do
  Let p_g be the most fit particle in s
  Let w_z be the parameter estimate for z within p_g
  Insert w_z into Θ_g
  if L(D|Θ_g) > L_best then
    L_best ← L(D|Θ_g)
    w_best ← w_z
  end if
end for
return w_best
```

5. EXPERIMENTS

To test the performance of our algorithms, several experiments were performed using data generated from networks in the Bayesian Network Repository [18]. We compared our algorithms to four competing approaches to parameter estimation: EM, Monte-Carlo EM (MCEM), Genetic Algorithm EM (GAEM), and Age-Layered EM (ALEM) [13, 14, 15, 21].

5.1 Methodology

For these experiments, we chose networks with gradually increasing numbers of parameters to evaluate the effect of network complexity on performance, in terms of log likelihood. For each network, we used forward sampling to generate 2000 data points and then removed the data for some

of the nodes, thereby simulating the presence of latent variables.

To evaluate the effect of latent variable structure on performance, we repeated the above sampling procedure for each network using two latent variable sets. The first set of data was generated using a latent variable architecture with a number overlapping Markov blankets, while the second was generated using latent variables with fewer overlapping Markov blankets. A cross-reference to the network information and latent variable configurations for the datasets is shown in Table 1. Datasets annotated with "O" were generated from networks with a greater number of overlapping latent variable structures, when compared to the datasets annotated with "I". We estimate the amount of overlap as the average number of nodes learned by each swarm:

$$\text{Overlap} = \frac{\sum_{s \in S} N(s)}{|S|}$$

where $N(s)$ is the number of nodes learned by swarm s. Note that since we are calculating Overlap based only on the swarms, this measure will differ on the same network depending on which variables are identified as being latent. We found that all of the networks with a greater number of overlapping latent variable structures had Overlap ≥ 3.

For our algorithm, six particles were assigned to each subswarm. For the other population based approaches, the total number of individuals was six times the number of incomplete nodes, to ensure that all population based algorithms had the same total number of individuals. 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 [5]. For MCEM, 400 samples were used during the expectation step.

To evaluate the effect of sampling from the dataset during fitness evaluation, we compared two versions of OSI. In the first, the entire training set was used during fitness evaluation, while the second version of OSI evaluated the parameters using a subset of the training data, containing $\frac{1}{3}$ of the original data points that were sampled at random during each fitness calculation. We denote the sampling based OSI algorithm as OSI-S.

To evaluate the performance of these algorithms, each data set was divided into training and testing data using a 5×2 cross-validation procedure, as recommended by Dietterich [3]. The log likelihoods for the best parameters found in each run were averaged over the runs for each algorithm. We compared the average log likelihoods of the algorithms using a paired t-test with a confidence interval of 95% to evaluate statistical significance.

5.2 Results

Table 2 shows the average log likelihoods for each algorithm and network configuration. Bold values indicate that the corresponding algorithm's performance is statistically significantly better than the competing algorithms. If two algorithms tied on the significance testing, both values are bolded.

On the networks with greater overlap, both OSI algorithms were statistically significantly better than all the other algorithms. OSI performed the best on Child-O, Alarm-O, Win95-O, and Hepar2-O networks while OSI-S did better on the Sachs-S and Insurance-O networks. MCEM was the worst performing algorithm on the Sachs-O and Hepar2-O networks. On the Child-O, Alarm-O, and Insurance-O, PSO was the worst performing algorithm while GAEM was the worst on the Win95-O network. Neither MCEM or GAEM outperform traditional EM on any of these networks.

On the networks with less overlap, OSI outperformed all other algorithms. However, while OSI and OSI-S were statistically significantly better than all the other algorithms on all networks, on the Alarm-I network, OSI was also statistically significantly than OSI-S. Additionally, OSI was only outperformed by OSI-S on the Insurance-I network. MCEM was the worst performing algorithm on the Sachs-I network while PSO performed the worst on the Child-I, Alarm-I, and Insurance-I network. For the Win95-I and Hepar-I networks, GAEM was the worst performing algorithm. EM was only outperformed by MCEM on the Hepar2-I network. On all other networks, EM outperformed MCEM and GAEM.

Finally, we observed that for three of the six networks, the gap between EM and OSI is much larger when the Markov blankets of the latent variables have greater overlap. Additionally, the performance of the OSI algorithms is worse when there was less overlap over the Markov blankets.

5.3 Discussion

The paired t-tests on the log likelihoods indicate that OSI performs better than the competing methods for all generated datasets. While PSO has consistently worse log likelihood than EM, OSI outperforms both EM and PSO on all datasets. Although ALEM managed to outperform traditional EM for most of the data-sets, the improvement was small compared that of OSI. Neither MCEM or GAEM outperform traditional EM, this is likely a result of the error introduced by the approximate Monte-Carlo based expectation step. The results also show that using a randomly sampled subset of the training data for fitness evaluation does not have a significant impact on the performance of OSI in terms of log likelihood for most datasets.

Although OSI outperforms EM and ALEM even when there is little overlap between the Markov blankets of latent variables, for half of the networks, the gap between the log likelihoods of OSI and EM were even larger when the latent variables had greater overlap between their Markov blankets. This is evidence that the improved performance obtained by OSI is due to the representation of each swarm being based on the Markov blankets and the corresponding competition that occurs between overlapping swarms. Recall that each variable is conditionally independent of all other variables in the network given its Markov blanket. By defining each variable's swarm to cover its Markov blanket, we ensure that the swarm learns the corresponding parameters for all variables upon which that variable may depend.

The difference in log-likelihood when overlap is varied is particularly large for the Sachs and Hepar2 networks. This may be because of the large difference in the overlap metric between the two configurations of the networks. However, we do not see similar results for the Win95 network, which has the most extreme difference in overlap between the two configurations. It appears that the effect of overlap on the performance of OSI could vary between networks, but further experiments must be performed to verify this claim and determine the effect of overlap on OSI's performance.

There are several additional advantages provided by our multi-swarm approach that may explain why our method outperforms competing approaches, even when there is little

Table 1: Network Statistics

Network	Params	Nodes	Label	Latent Variables	Overlap
Sachs	178	11	O	PKA, Raf, Erk	3.57
			I	PIP3, Raf, Erk	2.33
Child	230	20	O	LungParench, HypDistrib, HypoxiaInO2, ChestXray	3.00
			I	LVH, HypDistrib, Sick, CO2	2.11
Alarm	509	37	O	VENTLUNG, INTUBATION, SAO2, CATECHOL	3.20
			I	CO, VENTALV, LVEDVOLUME, VENTTUBE	2.64
Win95	574	76	O	NtGrbld, LclGrbld, DSLCLOK, AppData, DSNTOK	3.89
			I	TTOK, PSGRAPHIC, CmpltPgPrntd, AppDtGnTm, DSLCLOK	2.33
Insurance	984	27	O	CarValue, ThisCarCost, OtherCarCost, VehicleYear	3.67
			I	ThisCarCost, SeniorTrain, DrivQuality, Cushioning	2.75
Hepar2	1453	70	O	Obesity, Steatosis, RHepatitis, Hepatomegaly	3.89
			I	Obesity, Joints, Encephalopathy, Injections	2.40

Table 2: Comparison against other approaches

Network	EM	MCEM	GAEM	ALEM	PSO	OSI	OSI-S
Sachs-O	-5700.67	-7049.28	-6664.70	-5602.99	-6804.89	**-854.82**	**-834.18**
Child-O	-10598.31	-11706.62	-11477.23	-10519.32	-11751.61	**-3534.98**	**-3574.44**
Alarm-O	-11053.01	-13172.76	-13043.30	-10572.05	-13365.95	**-4797.21**	**-4811.59**
Win95-O	-8951.64	-9387.82	-9505.28	-8786.31	-5542.32	**-5194.31**	**-5814.35**
Insurance-O	-12316.01	-13774.30	-14577.78	-12209.57	-15622.23	**-7575.05**	**-7555.93**
Hepar2-O	-33329.08	-34504.95	-33970.11	-32704.60	-34105.81	**-19213.89**	**-19337.85**
Network	**EM**	**MCEM**	**GAEM**	**ALEM**	**PSO**	**OSI**	**OSI-S**
Sachs-I	-5168.45	−6515.06	-5946.70	-5173.73	-5979.34	**-3110.52**	**-3545.88**
Child-I	-10966.85	-11263.68	-11250.29	-10715.44	-11342.20	**-5118.27**	**-5499.05**
Alarm-I	-11037.85	-13585.07	-13541.31	-10749.36	-13995.45	**-4396.54**	-4761.65
Win95-I	-9294.90	-9792.64	-9861.66	-9081.66	-8630.77	**-5357.83**	**-5490.31**
Insurance-I	-12554.39	-14234.46	-14014.89	-12375.49	−15504.62	**-8942.22**	**-8855.79**
Hepar2-I	-31249.45	-31214.12	-31261.57	-31195.40	-31006.87	**-26198.50**	**-26719.78**

overlap between the Markov blankets of latent nodes. First, since the sub-swarms maintain independence, each swarm can explore a different region of the search space. Second, by splitting the swarms over the nodes of the network, we reduce the possibility of neglecting a potentially good parameter for a specific component of the global network's complete parameter set.

Also, since several swarms learn the parameters for a single variable, OSI allows for greater exploration of the search space and the competition between swarms ensures that the best parameter estimates found by the swarms are used in the global network. It is likely that this increased exploration contributes to the improved performance of OSI when compared to the EM algorithm, which is known to converge to local optima.

While these results are encouraging, more work must be done to empirically verify the effect of conditional dependencies on the optimal overlap structure. One area of improvement is the sampling procedure. It is possible that our the sampling procedure could result in some data points existing in both the training and testing data, thus introducing a bias into our results, but the number of times this occurred was small. In addition, since our goal is to parameterize the distribution with latent variables, one could argue that removing these data points would also bias the results negatively due to under-sampling. In addition, we are currently researching the effect of applying the OSI framework to other stochastic search techniques such as genetic algorithms, differential evolution, and simulated annealing to determine if the benefit of overlap is algorithm specific.

6. CONCLUSIONS

We have presented a swarm-based method for parameter estimation in Bayesian networks. In our approach, a swarm is associated with each hidden/latent variable in the network, and that swarm learns the parameters for its corresponding node. We compared our algorithm to several other approaches to parameter estimation, including a traditional single-swarm PSO. Our results indicate that OSI significantly outperforms the competing approaches on all of the datasets studied in terms of the log likelihood of the data given the parameters. Additionally, we found that, by sampling from the training data during fitness evaluation, we can reduce the computational burden of the algorithm without impacting the quality of learned parameters.

For future work, we will investigate the affect OSI parameters, such as the number particles per sub-swarm and how often competition is performed, have on OSI's performance. Additionally, the traditional OSI algorithm uses a sharing step in which the global solution is used to seed val-

ues in other sub-swarms. In this work, the sharing step was left out because it was found to dramatically decrease OSI's performance. More work is needed to investigate why OSI's performance decreased when OSI performed sharing.

Other areas of future work include investigating the existence of optimal swarm overlap structures. In the work presented here, we used the Markov blanket to derive OSI's sub-swarm architecture. We conjecture that, for many optimization problems, optimal OSI sub-swarm architectures exist related to the conditional dependence/epistatic properties of the underlying fitness landscape. We are exploring theoretical avenues to support this claim and are developing procedures to derive the swarm architectures based on a transformed representation of the optimization problem whereby we can determine each variable's Markov blanket. This would then allow us to know how to apply OSI to a large spectrum of optimization problems.

Finally, we are exploring how to use OSI to learn the structure of Bayesian networks. In all of the work presented in this paper, we assumed that the structure of the network was given; however, in many scenarios, the structure of the Bayesian network is unknown and must be learned from data. We plan to adapt OSI to the Bayesian structure learning problem, after which we can then use the work presented here as a way to learn the parameters for that structure. This will give us a OSI algorithm to learn a Bayesian network given only the data.

7. REFERENCES

[1] T. Belding. The distributed genetic algorithm revisited. In *Proceedings of the International Conference on Genetic Algorithms*, pages 114–121, 1995.

[2] G. Celeux and J. Diebolt. A stochastic approximation type EM algorithm for the mixture problem. *Stochastics: An International Journal of Probability and Stochastic Processes*, 41(1-2):119–134, 1992.

[3] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

[4] 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*, pages 39–43, 1995.

[5] R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of IEEE Congress on Evolutionary Computation*, volume 1, pages 84–88, 2000.

[6] G. Elidan and N. Friedman. Learning hidden variable networks: The information bottleneck approach. In *Journal of Machine Learning Research*, pages 81–127, 2005.

[7] G. Elidan, M. Ninio, N. Friedman, and D. Shuurmans. Data perturbation for escaping local maxima in learning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 132–139, 2002.

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

[9] N. Fortier, J. Sheppard, and S. Strasser. Learning bayesian classifiers using overlapping swarm intelligence. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 1–8, 2014.

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

[11] D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[12] B. K. Haberman and J. W. Sheppard. Overlapping particle swarms for energy-efficient routing in sensor networks. *Wireless Networks*, 18(4):351–363, 2012.

[13] W. Jank. The EM algorithm, its randomized implementation and global optimization: Some challenges and opportunities for operations research. In *Perspectives in operations research*, pages 367–392. Springer, 2006.

[14] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[15] O. J. Mengshoel, A. Saluja, and P. Sundararajan. Age-layered expectation maximization for parameter learning in bayesian networks. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, pages 984–992, 2012.

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

[17] K. M. Salama and A. A. Freitas. Learning bayesian network classifiers using ant colony optimization. *Swarm Intelligence*, 7(2-3):229–254, 2013.

[18] M. Scutari. Bayesian network repository. http://www.bnlearn.com/bnrepository/, 2012.

[19] 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*, pages 434–439. University of Michigan, 1989.

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

[21] G. C. Wei and M. A. Tanner. A monte carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 85(411):699–704, 1990.

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

[23] D. Whitley and T. Starkweather. Genitor ii: A distributed genetic algorithm. *Journal of Experimental & Theoretical Artificial Intelligence*, 2(3):189–214, 1990.