INFERENCE AND LEARNING IN BAYESIAN NETWORKS USING OVERLAPPING SWARM INTELLIGENCE

by

Nathan Lee Fortier

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

April, 2015

© COPYRIGHT

by

Nathan Lee Fortier

2015

All Rights Reserved

DEDICATION

To my loving wife, Amber; for her love, devotion, and endless support

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. John Sheppard for his constant support during my time as a graduate student here at Montana State University. His encouragement has pushed me to excel and has helped me mature as a researcher. This research would not have been possible without his support.

I would also like to thank to my committee members, Clem Izurieta, Mike Wittie, John Paxton, and Peter Tillack, for their helpful comments and advice.

I would like to thank the members of the Numerical Intelligent Systems Laboratory at Montana State University for their comments and advice during the development of this work. I would also like to thank Shane Strasser and Karthik Ganesan Pillai for their ideas during the development of this research.

On a more personal note, I would like to thank my wife and family for supporting me during the development of this work. My beautiful wife, Amber, has stood by me, giving me strength and encouragement whenever I needed it. My parents, Tish and Randy Fortier, encouraged me to develop a scientific mind from an early age and have always supported me in my endeavors.

TABLE OF CONTENTS

TABLE OF CONTENTS CONTINUED

TABLE OF CONTENTS CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF ALGORITHMS

Algorithm Page

ABSTRACT

While Bayesian networks provide a useful tool for reasoning under uncertainty, learning the structure of these networks and performing inference over them is NP-Hard. We propose several heuristic algorithms to address the problems of inference, structure learning, and parameter estimation in Bayesian networks. The proposed algorithms are based on Overlapping Swarm intelligence, a modification of particle swarm optimization in which a problem is broken into overlapping subproblems and a swarm is assigned to each subproblem. The algorithm maintains a global solution that is used for fitness evaluation, and is updated periodically through a competition mechanism. We describe how the problems of inference, structure learning, and parameter estimation can be broken into subproblems, and provide communication and competition mechanisms that allow swarms to share information about learned solutions. We also present a distributed alternative to Overlapping Swarm Intelligence that does not require a global network for fitness evaluation.

For the problems of full and partial abductive inference, a swarm is assigned to each relevant node in the network. Each swarm learns the relevant state assignments associated with the Markov blanket for its corresponding node. In our approach to parameter estimation, a swarm is associated with each node in the network that corresponds to either a latent variable or a child of a latent variable. Each node's corresponding swarm learns the parameters associated with that node's Markov blanket. We also apply Overlapping Swarm Intelligence to several variations of the structure learning problem: learning Bayesian classifiers, learning Bayesian networks with complete data, and learning Bayesian networks with latent variables. For each problem, a swarm is associated with each node in the network.

This work makes a number of contributions relating to the advancement of Overlapping Swarm Intelligence as a general optimization technique. We demonstrate the applicability of Overlapping Swarm Intelligence to both discrete and continuous optimization problems. We also examine the effect of the swarm architecture and degree of overlap on algorithm performance. The experiments presented here demonstrate that, while the sub-swarm architecture affects algorithm performance, Overlapping Swarm Intelligence continues to perform well even when there is little overlap between the swarms.

CHAPTER 1

INTRODUCTION

Bayesian networks have proven to be useful tools for reasoning under uncertainty and have been applied to 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 authors have used Bayesian networks for classification in medical diagnostics, music identification, and other areas [5, 6, 7]. There are many reasons for the appeal of Bayesian networks in these domains. First, they are directed graphical models that provide an intuitive representation of cause-effect relationships. Second, they handle uncertainty using well established probability theory. Finally, they can be used to represent both direct and indirect conditional dependencies.

However, using Bayesian networks in any field requires the construction of an effective model. While Bayesian networks can often be constructed by experts, in many cases the problem of defining a Bayesian network is too complex for manual construction. For these cases, the structure and parameters of the network can be learned from data.

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. 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 and the associated classes. When complete data is present, many of

the scoring functions used to evaluate Bayesian network structure can be decomposed into a product of terms, one for each family (each node and its parents). However, in practice, the 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, latent variables complicate the problems of parameter estimation and structure learning. When latent variables are introduced to a Bayesian network, the probability of the data given the model no longer decomposes into a product of terms and cannot be computed efficiently [12]. Also, since the likelihood of the parameters is no longer decomposable, conditional dependencies exist between the optimal parameters and structure for the network's variables. Because these latent variables are not recorded in the training data, parameters must be estimated for distributions that include these variables. Such parameter estimation may also be required when the training data is incomplete.

Once a model has been constructed, inference must be performed to answer queries. In some cases, inference involves computing the probability associated with the states of one or more variables in the model given a set of observations, while in other cases inference may involve discovering the most probable state assignments for a set of variables in the network given the evidence. The latter problem is referred to as abductive inference. In some cases, we may be interested in ranking the k most probable assignments to be examined by some other process. Thus the k-MPE task is to find and return these k most probable assignments. In [13], it was shown that abductive inference for Bayesian networks is NP-hard. Because of this, much research has been done to explore the possibilities of obtaining partial or approximate

solutions to the problem. However in [14], it was shown that even the problem of finding a constant factor approximation of the k-MPE is NP-hard.

1.1 Contributions

This work presents novel approximation algorithms for the problems of Bayesian classifier learning, general Bayesian structure learning, latent variable learning, parameter estimation, and abductive inference in Bayesian networks. These algorithms are based on the overlapping swarm intelligence (OSI) framework, first introduced by Haberman and Sheppard [15]. For each of these problems, a particle swarm optimizer is split into multiple swarms, and a swarm is assigned to each node in the network. For all problems, a global solution is maintained and used for fitness evaluation. Swarms with overlapping solutions compete for inclusion in the global solution.

We also present an algorithm, called distributed overlapping swarm intelligence (DOSI), that eliminates the need for a global solution vector. We demonstrate empirically that this distributed algorithm is often competitive with OSI and we define the necessary conditions under which the distributed swarms will reach consensus.

We examine a variety of different swarm architectures for the various problems. In the case of abductive inference and parameter estimation, each swarm learns the state/parameters associated with its node's Markov blanket. However, for the structure learning problems, assigning each swarm to a fixed Markov blanket structure will not be possible since the network structure is unknown. In the case of general structure learning from complete data, each swarm learns the parents for its corresponding node. This swarm architecture does not exhibit any overlap, thus competition between the swarms will be unnecessary. For the learning of Bayesian classifiers, each swarm learns both the parents and children for its associated node

and swarms compete where these structures overlap. Finally, for the problem of structure learning in the presence of latent variables, each swarm learns the parents for the nodes of its Markov blanket within the current global best structure. In this architecture, the set of nodes learned by a swarm will change dynamically as the global structure improves.

Examining the performance of OSI on problems related to Bayesian networks allows us to begin testing important properties of the framework. The explicit conditional independence properties of Bayesian networks provide a convenient setting for testing the effect of conditional dependencies on effective OSI overlap structures, and the problem of structure learning with latent variables allows us to explore how OSI may be applied to problems when conditional dependencies are unknown. The results presented for the continuous parameter estimation problem and the discrete structure learning and inference problems provide strong evidence that the effectiveness of OSI is not tied to a specific problem domain or search space. Finally, we provide both an empirical and theoretical analysis of OSI's computational complexity, as compared to traditional PSO.

We also present and test a number of hypotheses. First, OSI will outperform PSO in terms of solution quality for most experiments and will never perform worse than single-swarm PSO. For full and partial abductive inference, this means that the the state assignments found by OSI and DOSI will have higher log-likelihood than those found by DMVPSO for most experiments on complex networks. For parameter estimation, this means that the likelihood of the data given the model will be higher for parameters learned by OSI as compared to those learned by single-swarm PSO. When learning the structure of Bayesian classifiers, the hypothesis will be supported if the models learned by variants of OSI have higher classification accuracy than those learned by DMVPSO for most datasets. For the task of generative structure

learning, both with and without latent variables, the hypothesis will be supported if the likelihood of the data given the model is higher for models learned by OSI as compared to those learned by single-swarm PSO for the majority of datasets.

Our second hypothesis is that OSI and DOSI will tie statistically in terms of log-likelihood for most experiments. For the problems of full and partial abductive inference, this hypothesis will be supported if there is no significant difference between the log-likelihoods of assignments found by OSI and the log-likelihoods of assignments found by DOSI, for most networks.

Our third hypothesis is that the Markov blanket overlap structure will not be outperformed by any other overlap architecture, and will often outperform alternative architectures. For the abductive inference problems, this hypothesis will be supported if the average log-likelihood for state assignments found by OSI when using the alternative overlap architectures is never significantly greater than the average log-likelihood for state assignments found by OSI when using the Markov blanket architectures. Additionally, this hypothesis predicts that OSI will often find state assignments with higher log-likelihood when using the Markov blanket architecture, when compared to the state assignments found when using the alternative architectures. For the problem of structure learning with latent variables, this hypothesis will be supported if OSI often learn models with higher log-likelihood when using the dynamic Markov blanket architecture, as compared to the models learned by the static family-based architecture.

This work describes these algorithms, presents results comparing OSI to other commonly used approaches, and includes a discussion of the meaning of these results in the context of specific problems and in terms of the OSI framework generally. Through the problems of inference, learning, and parameter estimation in Bayesian

networks, we show that OSI is a versatile method that is applicable to a wide variety of problems.

1.2 Overview

The primary contributions of this work are the development of new methods, based on Overlapping Swarm Intelligence, that are able to perform inference on Bayesian Networks and learn the structure and parameters of Bayesian networks more effectively than existing approaches. The remainder of this thesis is organized as follows:

In Chapter 2, we review the background work related to this thesis. This includes an overview of Bayesian networks and an introduction to traditional methods for Bayesian network inference and learning. We also describe the Particle Swarm Optimization algorithm for both discrete and continuous optimization problems.

In Chapter 3, we review the literature related to this thesis. In this chapter, we describe the existing algorithms in the areas of distributed soft computing and optimization. We also review existing approaches to inference, parameter estimation, and structure learning in Bayesian networks.

In Chapter 4, we describe our algorithms for full abductive inference in Bayesian networks. We present both centralized and distributed versions of OSI for this problem. We compare OSI and DOSI to competing approaches for full abductive inference in terms of both solution quality and computational complexity, and we evaluate effect of the various components of OSI in terms of log-likelihood of the solutions found.

In Chapter 5, we present our algorithms for partial abductive inference in Bayesian networks. We describe both centralized and distributed versions of OSI for partial abductive inference and we compare OSI to other approaches for partial abductive inference in terms of both solution quality and computational complexity.

In Chapter 6, we describe an algorithm for parameter estimation in Bayesian networks based on OSI. In this chapter we also present a new single-swarm PSO algorithm for parameter estimation, and OSI is compared to this traditional PSO algorithm along with several other algorithms used for parameter estimation. Here we also present results evaluating the effect of overlap on OSI performance.

In Chapter 7, we present three methods for Bayesian network structure learning based on OSI. The first is an algorithm for learning the structure of the attribute nodes in a Bayesian classifier. The second is an algorithm for generative Bayesian network structure learning. We compare this algorithm to a traditional PSO and to a commonly used algorithm for generative structure learning with complete data. The third is an algorithm for learning Bayesian network structure with latent variables, which uses a dynamic overlap architecture. We compare this algorithm to structural Expectation Maximization, and a traditional single-swarm PSO.

In Chapter 8, we conclude with a summary of contributions, an overview of potential future work to expand upon the research presented here, and a survey of published research directly related to this work.

CHAPTER 2

BACKGROUND

When reasoning in any domain, complete knowledge is almost always impossible to obtain and many facts about a system are often left unknown. For this reason, numerous methods have been developed to summarize and quantify the uncertainty of a system by specifying numeric measures of uncertainty. Bayesian networks are one of the most popular models used to quantify such uncertainty. These models have been used in areas such as information retrieval [16], risk analysis [17, 18, 19], bioinformatics [20, 21, 22], and medicine [23]. However, many difficult problems must be solved to use Bayesian networks effectively. These problems include, inference, structure learning, and parameter estimation.

2.1 Bayesian Networks

Prior to the introduction of Bayesian networks, full joint distribution tables were commonly used for probabilistic reasoning tasks. However, even in the case of binary valued variables, a full joint distribution requires specifying an exponential number of parameters. Such distributions are difficult to manipulate computationally, and it is nearly impossible for a human expert to specify the large number of required parameters. Bayesian networks address this problem by providing a compact representation of the joint distribution that exploits the conditional independence properties of the distribution [24].

More formally, a Bayesian network is a directed acyclic graph that encodes a joint probability distribution over a set of random variables $\mathbf{X} = \{X_1, ...X_n\}$, where each

variable can assume one of several mutually exclusive values [12]. In a Bayesian network (BN), 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 any given node X in the network, the conditional distribution $P(X|\mathrm{Pa}(X))$ is said to be a factor ϕ over $\{X\} \cup \mathrm{Pa}(X)$. The set of variables used to specify a factor ϕ is called the scope of ϕ. For any set of random variables in the network, the probability of any state of the joint distribution can be computed using the chain rule.

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

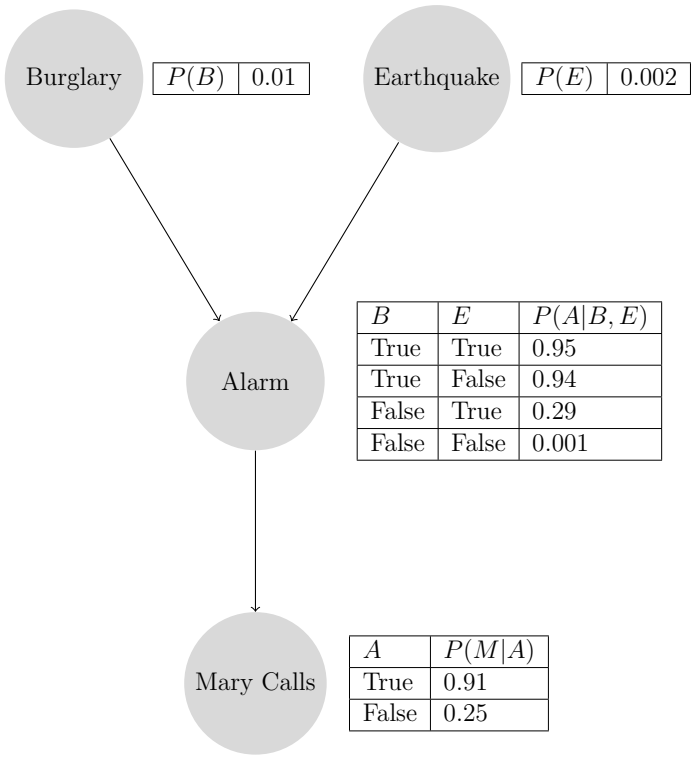
Using the local distributions specified by the BN, the joint distribution can be represented equivalently as

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

Given a set of variables X, Y, Z associated with a Bayesian network G, X is conditionally independent of Y given Z in the distribution defined by G if G satisfies $P(X, Y|Z) = P(X|Z)P(Y|Z)$ for all values $x \in \mathrm{Val}(X)$, $y \in \mathrm{Val}(Y)$, and $z \in \mathrm{Val}(Z)$. 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.

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

Figure 2.1: Alarm network



B	E	$P(A\|B,E)$
True	True	0.95
True	False	0.94
False	True	0.29
False	False	0.001

A	$P(M\|A)$
True	0.91
False	0.25

We exploit Markov blankets and their conditional independence properties in the particle representation used for abductive inference, and parameter estimation. While the relationship between a node and its children/parents seems intuitive, the relationship between a node and its children's parents is less obvious. This dependency is the result of interactions between the various causes of the same effect.

To illustrate using a Bayesian network, suppose we have installed a new alarm system in our home. Although this alarm system will detect a burglary reliably, it may also be set off by an earthquake. Also, suppose a friend, Mary, has promised to call whenever she hears the alarm. A Bayesian network for this system is presented in Figure 2.1. The network structure shows that earthquakes and burglaries directly effect the alarm but the probability of Mary calling only depends on the alarm.

Suppose we know that the alarm went off earlier today and we would like to know the probability of a burglary. While we know the probability distribution for the alarm given a burglary and an earthquake, we do not have a distribution in terms of our two variables of interest (i.e. alarm and burglary). That is, we have an extra variable, earthquake, that must be eliminated from the distribution. Conveniently we can eliminate any variable by summing terms from the full joint distribution. Specifically, the variable earthquake can be eliminated by a process called summing-out or marginalization as follows:

$$P(A,B) = \sum_{e \in E} P(A,B,e) = \sum_{e \in E} P(A|B,e)P(B)P(e) = P(B) \sum_{e \in E} P(A|B,e)P(e)$$

Using this process, along with the definition of conditional probability, we can compute following conditional probability of a burglary given the alarm:

$$P(B|A) = \frac{P(A,B)}{P(A)} = \frac{P(B) \sum_{e \in E} P(A|B,e)P(e)}{\sum_{b \in B} P(b) \sum_{e \in E} P(A|b,e)P(e)} = 0.86$$

Now suppose that we later learn that there was an earthquake in the area. The updated conditional probability of a burglary is now given by

$$P(B|A,E) = \frac{P(B)P(A|B,E)P(E)}{\sum_{b \in B} P(A|b,E)P(E)P(b)} = 0.03$$

Essentially, we have provided an alternative explanation for the Alarm being triggered, thereby reducing the probability that a burglary was the cause of the alarm. In other words, we have explained away the alarm via the observation of an earthquake. Explaining away is an example of a reasoning pattern called inter-causal reasoning, in which the various causes of the same effect interact.

2.2 Exact Inference in Bayesian Networks

In Bayesian networks, inference is the process of answering probabilistic queries about the network. In standard inference the probability of the state of one or more variables in the network is queried, while in abductive inference the goal of the query is to find the most probable state assignment for one or more variables.

2.2.1 Variable Elimination

Variable elimination is an exact algorithm for computing $P(X|\mathbf{E})$ in Bayesian networks, where X is an unobserved variable in the network and $\mathbf{E}$ is a set of observed evidence variables in the network. The pseudocode for variable elimination is shown in Algorithm 2.1. The primary operation used during variable elimination is that of marginalizing out variables from the distribution.

Let $\mathbf{\Phi}$ be the factors parameterizing some Bayesian network G with a set of variables $\mathbf{X}$. Let $Z \in \mathbf{X}$ be some variable and let $\mathbf{\Phi}' = \{\phi \in \mathbf{\Phi} : Z \in \text{Scope}[\phi]\}$. The factor marginalization of the variable Z, can be defined as a new factor ψ such that:

$$\psi(Z) = \sum_{Z} \prod_{\phi \in \mathbf{\Phi}'} \phi \tag{2.1}$$

When the variable, Z, has been marginalized out using the above process, we say that the variable has been eliminated.

For a given query $P(Y|\mathbf{E})$, variable elimination begins by eliminating the evidence variables $\mathbf{E}$. This is done by iterating over each local distribution $\phi \in \mathbf{\Phi}$ and removing entries that are inconsistent with the evidence. After inconsistencies are removed, the algorithm removes all references to evidence variables from the distributions. Next, the algorithm eliminates all variables that are not evidence or query variables through

Algorithm 2.1 Variable Elimination for computing $P(Y|E)$

Procedure Variable-Elim(
  $\mathbf{X}$ // Set of variables in the network
  Y // Query variable
  $\mathbf{E}$ // Evidence
  $\mathbf{\Phi}$ // Set of factors in the network
)

1: Eliminate evidence variables from $\mathbf{\Phi}$
2: $\mathbf{Z} \leftarrow \mathbf{X} - \{Y\} - \mathbf{E}$
3: **for** $Z_i \in \mathbf{Z}$ **do**
4: $\quad \mathbf{\Phi}' \leftarrow \{\phi \in \mathbf{\Phi} : Z_i \in \text{Scope}[\phi]\}$
5: $\quad \mathbf{\Phi}'' \leftarrow \mathbf{\Phi} - \mathbf{\Phi}'$
6: $\quad \psi(Z_i) = \sum_{Z_i} \prod_{\phi \in \mathbf{\Phi}'} \phi$
7: $\quad \mathbf{\Phi} \leftarrow \mathbf{\Phi}'' \cup \{\psi\}$
8: **end for**
9: $\phi^* \leftarrow \prod_{\phi \in \mathbf{\Phi}} \phi$
10: $\alpha \leftarrow \sum_{y \in \text{Val}(Y)} \phi^*(y)$
11: **return** $\phi^* \alpha^{-1}$

marginalization. During this process the algorithm iterates over this set of variables and eliminates each one as described in equation 2.1. After eliminating all irrelevant variables, the algorithm will compute the normalizing coefficient α as follows:

$$\alpha = \sum_{y \in Y} \phi^*(y)$$

Once all variables have been eliminated, the algorithm computes the posterior probability of the query variables as:

$$P(Y|\mathbf{E}) = \frac{\phi^*}{\alpha}$$

The time complexity of variable elimination is tied to the order in which the variables are eliminated. While the problem of finding an optimal variable ordering is known to be NP-complete [12], several heuristics have been proposed for choosing

a variable order. These include choosing variables for elimination with a minimum number of neighbors and choosing variables that will minimize the number of edges that will be added to the graph following elimination.

2.2.2 Clique Tree Propagation

Clique tree propagation is an algorithm for exact inference in probabilistic graphical models that is more efficient than variable elimination when performing inference over multiple variables. The algorithm relies on a data structure called a clique tree, which is a special type of clique graph. A clique graph provides a graphical representation of the factor-manipulation process in variable elimination. A Bayesian network is said to be moralized if an edge exists between all non-adjacent parents that share a child. Each node in the clique graph corresponds to a sub-clique in the triangulation of the moralized Bayesian network. Clique trees exhibit the running intersection property. A clique graph satisfies the running intersection property if, for each pair of cliques $\mathbf{X}$ and $\mathbf{Y}$ with intersection $\mathbf{Z}$, all cliques on the path between $\mathbf{X}$ and $\mathbf{Y}$ contain $\mathbf{Z}$. A clique tree can be constructed from a Bayesian Network as follows [12]:

- Moralize the graph by adding an edge between all non-adjacent parents of the same child
- Triangulate the moralized graph by inserting edges in cycles longer than length 3 so that all shortest cycles are of length 3.
- Find all maximal sub-cliques in the moralized triangulated graph and make each clique a node in the clique graph (Note: in general, the maximal sub-clique problem is intractable; however, maximal sub-cliques can be found in polynomial time in perfect graphs, i.e., where the graphs clique number is equal

Algorithm 2.2 SP-Message(i, j)

Procedure SP-Message(
i // Index of sending Clique
j // Index of receiving clique
)

1: $\psi_i(\mathbf{C}_i) \leftarrow \psi_i \prod_{k \in Nb_i - \{j\}} \delta_{k \to j}$ // compute clique potential
2: $\tau(S_{i,j}) \leftarrow \sum_{\mathbf{C}_i - S_{i,j}} \psi_i(\mathbf{C}_i)$ // compute resulting factor to be sent as a message
3: **return** $\tau(S_{i,j})$

to its chromatic number. Bayesian networks that have been moralized and triangulated are perfect graphs.)

- Add an edge between intersecting cliques in the clique graph labeled with their sepset $S_{i,j} = \mathbf{C}_i \cap \mathbf{C}_j$
- Let the weight of an edge in the clique graph be the cardinality of its sepset $|S_{i,j}|$
- Obtain the clique tree by finding the maximum spanning tree of the clique graph

Once constructed, a clique tree can be used to perform inference in Bayesian networks through a process called message passing. Each clique in the clique tree takes incoming messages from its neighbors, multiplies them, sums out one or more variables (similar to variable elimination), and sends the newly computed message to a neighboring clique. Algorithm 2.2 describes the message passing algorithm.

When computing a message, the clique $\mathbf{C}_i$ multiplies all incoming messages $\delta_{i \to j}$ from its other neighbors N_i with its initial clique potential ψ_i to obtain a new factor $\psi_i(\mathbf{C}_i)$. It then sums out all variables except those in the intersection ($S_{i,j}$) of $\mathbf{C}_i$ and $\mathbf{C}_j$ and sends the resulting factor $\tau(S_{i,j})$ as a message to $\mathbf{C}_j$. Using these messages a clique tree can compute the probabilities of all variables in the network without need-

Algorithm 2.3 Clique Tree Calibration

Procedure CTree-Calibrate(
  $\mathbf{\Phi}$ // Set of factors in the network
  T // Clique tree for network
)

1: Initialize Cliques
2: // Perform message passing
3: **while** exist i, j such that i is ready to transmit to j **do**
4: $\quad \delta_{i \to j}(S_{i,j}) \leftarrow \text{SP-Message}(i, j)$
5: **end while**
6: // Compute beliefs for all cliques
7: **for** each clique i **do**
8: $\quad \beta_i \leftarrow \psi_i \prod_{k \in \text{Nb}_i} \delta_{k \to i}$
9: **end for**
10: **return** $\{\beta_i\}$

lessly recomputing the same sum-product calculations multiple times. This process is called clique tree calibration and is presented in Algorithm 2.3.

The clique tree calibration algorithm shown above is defined as an asynchronous algorithm in which each clique sends a message to a neighboring clique as soon as it is ready. This algorithm is equivalent to a more synchronized process consisting of an upward pass and a downward pass. During the upward pass the algorithm picks a root clique and sends all messages toward the root. During the downward pass the root sends the appropriate messages to its children. This continues until the leaves of the tree are reached. Once the message passing process is complete, the beliefs β_i for all cliques are computed by multiplying the initial potential with each of the incoming messages. The desired posterior for any variable can then be determined by summing out the unwanted variables in a clique containing the target variable.

The primary advantage of the clique tree algorithm is that it calculates the posterior probability of all variables in the Bayesian Network using only twice the computation of the upward pass on the same tree. In other words, once the tree is calibrated,

it is not necessary to recalculate the posteriors of any of the variables in the network, unless new evidence is introduced.

2.2.3 Abductive Inference

Another type of inference used to query a Bayesian network is abductive inference. Abductive inference is the problem of finding the maximum a posteriori (MAP) probability state of the variables of a network, given a set of evidence variables and their corresponding state. This problem is often referred to as the k-Most Probable Explanation (k-MPE) problem.

If we let $\mathbf{X}_U = \mathbf{X} \backslash \mathbf{X}_O$, where $\mathbf{X}$ denotes the variables in the network and $\mathbf{X}_O$ denotes the set of evidence variables in the network, the problem of abductive inference is to find the most probable state assignment to the variables in $\mathbf{X}_U$ given the evidence $\mathbf{X}_O = x_O$:

$$MPE(\mathbf{X}_U, x_O) = \arg\max_{x_u \in \mathbf{X}_U} P(x_u | x_O)$$

In [25], it was shown that abductive inference for Bayesian networks is NP-hard. Because of this, much research has been done to explore the possibilities of obtaining approximate solutions to the problem of full abductive inference. However in [14] it was shown that even the problem of finding a constant factor approximation for abductive inference is NP-hard.

Partial abductive inference is the problem of finding the k most probable state assignments for a subset of the variables $\mathbf{X}_E \subset \mathbf{X}_U$, known as the explanation set:

$$\begin{aligned} \text{PAI}(\mathbf{X}_E, x_O) &= \arg\max_{x_E \in \mathbf{X}_E} P(x_E | x_O) \\ &= \arg\max_{x_E \in \mathbf{X}_E} \sum_{x_R \in \mathbf{X}_R} P(x_E, x_R | x_O) \end{aligned}$$

where $\mathbf{X}_R = \mathbf{X}_U \backslash \mathbf{X}_E$. This problem is more useful than general abductive inference in most applications since it allows for the selection of only the relevant variables as the explanation set. However, Park proved that partial abductive inference is an even more complex problem than full abductive inference [26]. Specifically, Park showed that partial abductive inference is NP^{PP}-Complete. Unlike full abductive inference, when performing exact partial abductive inference, the variables in $\mathbf{X}_R$ must be marginalized out. This additional marginalization process increases the time complexity of partial abductive inference when compared to full abductive inference.

An exact algorithm capable of finding the MPE called bucket elimination was proposed by Dechter [27]. While bucket elimination is applicable to many problems in Bayesian inference, we will discuss it in the context of abductive inference. The pseudocode for bucket elimination in the context of MPE is presented in Algorithm 2.4. The first step of bucket elimination is to partition the factors of the Bayesian network into groups called buckets. A bucket is associated with each variable, and the bucket of a particular variable contains the probability functions associated with that variable. The next step, called the backward phase, processes the buckets based on a predefined variable ordering. A bucket $\mathbf{\Phi}'$ is processed by multiplying all probability functions in the bucket and then eliminating the bucket's variable X_i using a process called factor maximization:

$$\psi(X_i) = \max_{X_i} \prod_{\phi \in \mathbf{\Phi}'} \phi$$

This process is identical to the process of factor marginalization in variable elimination, except that the sum is replaced with a max operator. The distribution obtained by processing a bucket is placed in an earlier bucket of its largest index variable, where the index of a variable is based on its position in the ordering.

Algorithm 2.4 Bucket Elimination

Procedure Bucket-Elim(
  Φ // Set of factors in the network
  E // Evidence
)

Populate buckets $\mathbf{bucket}_1, .., \mathbf{bucket}_n$
Place each observed variable $X_p = x_p \in \mathbf{E}$ in its bucket
Let $S_1, ..., S_j$ be the subset of variables on which factors are defined
for $p \leftarrow n$ to 1 **do**
  for each factor $\phi_1, ..., \phi_j \in \mathbf{bucket}_p$ **do**
    if the $\mathbf{bucket}_p$ contains $X_p = x_p$ **then**
      assign $X_p = x_p$ to each ϕ_i and put each in the appropriate bucket
    else
      $U_p \leftarrow \cup_{i=1}^{j} S_i - \{X_p\}$
      $\psi(X_p) \leftarrow \max_{X_p} \prod_{\phi \in \mathbf{\Phi}'} \phi$
      $x_p^o \leftarrow \arg\max_{X_p} \psi(X_p)$
      Add h_p to the bucket of largest index variable in U_p
    end if
  end for
end for
Assign values in the ordering using the recorded functions x_p^o in each bucket

This assignment is stored in the bucket associated with Z. Once all buckets are processed, the algorithm initiates a forwards phase to compute the most probable state assignment by assigning values along the ordering from X_1 to X_n. After a partial assignment has been obtained for the variables $\{X_1, ..., X_{i-1}\}$, a value x_i is appended to the assignment where:

$$x_i = \arg\max_{X_i} \psi(X_i)$$

The backward phase of the algorithm aggregates information regarding the most probable assignment to variables higher in the ordering. As with variable elimination, the complexity of the bucket elimination is highly sensitive to the variable ordering used by the algorithm.

2.3 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, the 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 dependencies in a structure match the data, and they search for a structure that maximizes this function. A common scoring method to evaluate Bayesian network structures is the probability of the data given the structure. It was shown in [28] that the probability of the data $\mathbf{D}$ given a candidate Bayesian network structure G can be computed as follows.

Let $\mathbf{X}$ be the set of n discrete variables in G, where each variable $X_i \in \mathbf{X}$ has r_i possible value assignments: $(x_1, ..., x_{r_i})$. Let $\mathbf{D}$ be a set of data containing m cases, where each case contains a value assignment for each variable in $\mathbf{X}$. Each variable $X_i \in \mathbf{X}$ has a set of parents, represented by $\text{Pa}(X_i)$. Let w_{ij} denote the j^{th} unique instantiation of $\text{Pa}(X_i)$ relative to $\mathbf{D}$. Define q_i to be the number of unique instantiations of $\text{Pa}(X_i)$ relative to $\mathbf{D}$. Let N_{ijk} be the number of cases in $\mathbf{D}$ for which the variable X_i has the value x_k and $\text{Pa}(X_i)$ is instantiated as w_{ij}. Let

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

Then

$$P(\mathbf{D}|B) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!. \tag{2.2}$$

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

$$L(\mathbf{D}|B) = \log P(\mathbf{D}|B). \tag{2.3}$$

Another scoring metric commonly used for Bayesian network model selection is the Bayesian Information Criterion (BIC) developed by Gideon E. Schwarz [29]. This metric introduces a penalty term for the number of parameters in the candidate model to control overfitting. The formula for the BIC [30] is as follows:

$$\text{BIC} = -2 \log P(\mathbf{D}|B) + p \log(m) \tag{2.4}$$

where p is the number of free parameters and m is the size of the dataset.

K2 is a heuristic search technique for Bayesian network structure learning proposed by Cooper and Herskovits [31]. The algorithm is a greedy method that relies on a pre-specified node ordering. The main loop of the K2 algorithm iterates over all nodes in the ordering. For each node X, K2 incrementally adds edges between X and the parent node Y that results in the greatest increase to the likelihood of the model. If no such parent can be found, then K2 stops adding parents to the node and moves to the next node in the ordering. A node X can be added as the parent of another node Y if and only if Y precedes X in the ordering.

The pseudocode for K2 is shown in Algorithm 2.5. Here $\text{Pred}(x)$ denotes the nodes preceding x in the ordering and the function $g(x, P)$ returns the probability of the data given the model when the variables in P are set as the parents of X. The algorithm has been shown to have a time complexity of $\text{O}(mn^4r)$ where n is the

Algorithm 2.5 K2 Structure Learning Algorithm

```
    Procedure K2(
        D    // the training data
        X    // set of variables in specified ordering
    )
 1: for each variable X_i ∈ X do
 2:     Pa(X_i) ← ∅
 3:     P_old ← g(X_i, Pa(X_i))
 4:     while Pa(X_i) < u do
 5:         z ← arg max_{z∈Pred(X_i)−Pa(X_i)} g(X_i, Pa(X_i) ∪ {z})
 6:         P_new ← g(X_i, Pa(X_i) ∪ {z})
 7:         if P_new > P_old then
 8:             P_old ← P_new
 9:             Pa(X_i) ← Pa(X_i) ∪ {z}
10:         else break
11:     end while
12: end for
```

number of nodes, r is the maximum number of value assignments for any variable, and $m = |\mathbf{D}|$.

The techniques described above implicitly assume that our goal is to learn the Bayesian network that best approximates the underlying joint probability distribution of the data. This approach is called generative learning because we are learning the model to generate all variables in the data, including both the variables that we aim to predict and the evidence variables used for prediction. However, we often want to learn a network that will perform well on a particular task, such as predicting the state of a specific node from evidence without concern for the underlying joint distribution. For this type of problem we most likely want to learn the model discriminatively, where the goal is to find the network that best represents the true conditional probability distribution of the prediction given the evidence.

2.4 Bayesian Network Classifier Learning

In classification a more specific type of Bayesian network structure is often learned. Given a classification problem consisting of a series of attributes $\mathbf{A}$ and a class label C, a Bayesian classifier is a Bayesian network in which a node is associated with a each attribute $A_i \in \mathbf{A}$ and the class C. Typically, an edge is placed in the structure from the class node to each of the attribute nodes; although this is not necessary. Additional relationships and latent variables can also be included in this structure. 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, where all attributes are assumed to be conditionally independent given the class, and there are no edges connecting any of the attribute nodes. In this model, given some observed attribute values $a_1, ..., a_n$, the probability of each class can be computed as:

$$P(C|a_1, ..., a_n) = \frac{P(C)\prod_{i=1}^{n} P(a_i|C)}{\sum_{c\in C} P(c)\prod_{i=1}^{n} P(a_i|c)}$$

Learning a naive Bayes classifier requires estimating the prior probability of the class $P(C)$ and the conditional probabilities of the attributes $P(a|C)$. The prior probability of the class can be computed as:

$$P(C) = \frac{\sum_{d\in\mathbf{D}} \mathbf{1}_d(C)}{|\mathbf{D}|} \tag{2.5}$$

where $\mathbf{1}_d$ is an indicator function for a data point d. The conditional probability of each attribute A can be computed as:

$$P(A|C) = \frac{\sum_{d \in \mathbf{D}} \mathbf{1}_d(A, C)}{\sum_{d \in \mathbf{D}} \mathbf{1}_d(C)} \tag{2.6}$$

An alternative to Naive Bayes that relaxes the independence assumption is the Tree Augmented Naive Bayes (TAN) classifier described in [32]. The TAN algorithm adds edges to the network that maximize the conditional mutual information between the connected feature nodes given the class. Conditional mutual information is defined as follows:

$$I(X, Y|C) = \sum_{x,y,c} P(x, y, c) \log \frac{P(x, y|c)}{P(x|c)P(y|c)} \tag{2.7}$$

where X and Y are the feature variables, C is the class variable, 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.

For the TAN model, the prior probabilities for the class and the conditional probabilities for the root node of the spanning tree can be computed as shown in equations

2.5 and 2.6 respectively. The conditional probabilities for each non-root node X given its parent Y and the class C can be computed as:

$$P(X|Y,C) = \frac{\sum_{d\in\mathbf{D}} \mathbf{1}_d(X,Y,C)}{\sum_{d\in\mathbf{D}} \mathbf{1}_d(Y,C)}$$

The TAN algorithm has been shown to have higher accuracy that Naive Bayes on many problems. This algorithm is based on the ChowLiu algorithm, which is an efficient method for learning second-order product approximations of joint distributions using mutual information [33].

2.5 Parameter Estimation and Learning with Latent Variables

When learning a Bayesian network structure from data, the parameters associated with each node's probability distribution will need to be estimated. If a variable and its parents are present in the data, we can estimate its parameters directly based on frequency. Given some data-set $\mathbf{D}$ we can estimate any joint probability over the observed variables as

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

where

$$z = \prod_i^n |X_i|$$

In the above equations, $\mathbf{1}_d$ is an indicator function for a data point d, $|X_i|$ is the number 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

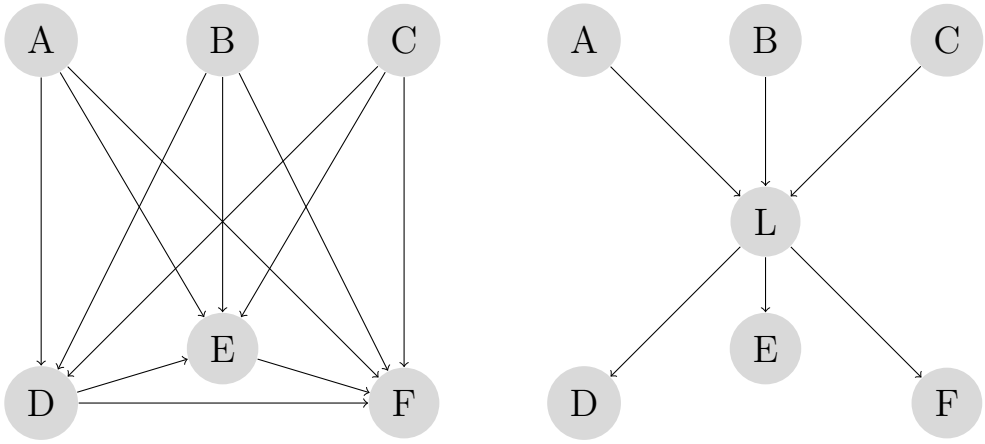
a subset of the observed variables as follows:

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

Although parameter estimation can be performed efficiently in the presence of complete data, 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. An example of this is shown in Figure 2.2. This figure is based on an example presented by Russell and Norvig [34], in which the nodes D, E, and F are symptoms of a disease, indicated by node L, while the nodes A, B, and C represent potential causes of the disease. If the disease variable L is not taken into account, it will appear that the symptoms D, E, and F interact. Additionally, the causes of the disease (A, B, and C), will appear to directly influence the the symptoms (D, E, and F). These apparent interactions are described by the Bayesian network in Figure 2.2(a). By introducing a latent variable L to represent the disease, we can explain many of the apparent interactions, thereby producing the Bayesian network shown in Figure 2.2(b).

Suppose that each variable in these networks has two states. Prior to introducing the latent variable L, the model shown in Figure 2.2(a) requires 118 parameters, but after the introduction of the latent variable L, the model only requires 34 parameters, as shown in 2.2(b).



(a) Model without latent variables (b) Model with latent variable

Figure 2.2: Latent variable example

A score $s(B : \mathbf{D})$ of a Bayesian network B is said to be decomposable if the score can be written as:

$$s(B : \mathbf{D}) = \sum_{X_i \in \mathbf{X}} \text{FamScore}(X_i | \text{Pa}(X_i) : \mathbf{D})$$

where $\text{FamScore}(X_i|\text{Pa}(X_i) : \mathbf{D})$ measures how the variables $\text{Pa}(X_i)$ serve as the parents of X_i. Score decomposability has dramatic implications when searching for structures that maximize some score. If a score is decomposable, then a local change in network will not change the score of other sub-structures and the computational overhead of evaluating different structures can be reduced.

Unfortunately, latent variables complicate the problem of parameter estimation. When latent variables are introduced to a Bayesian network, the marginal probability is no longer decomposable and cannot be computed efficiently [12]. This problem could be addressed by filling in the values for the missing variables, but this would require knowledge of the very parameters that we are estimating. Thus, parameter

estimation from incomplete data requires solving two optimization problems simultaneously: learning the parameters and estimating values for the latent variables in each data point.

Expectation maximization (EM) is a commonly used algorithm that provides a solution to this problem [35]. EM begins with an initially random set of estimated parameters and then alternates between two steps.

Expectation step: The algorithm uses the current parameters θ^t to estimate the expected sufficient statistics.

- For each data point d_i and each variable X, calculate the joint distribution $P(X, \mathrm{Pa}(X)|d_i, \theta^t)$
- Estimate the sufficient statistics for each assignment x and $\mathrm{Pa}(x)$ as

$$M_{\theta^t}[x, \mathrm{Pa}(x)] = \sum_i P(X, \mathrm{Pa}(X)|d_i, \theta^t)$$

Maximization step: The algorithm performs maximum likelihood estimation relative to the expected sufficient statistics to estimate the new parameters.

$$\theta^{t+1}_{x|\mathrm{Pa}(x)} = \frac{M_{\theta^t}[x, \mathrm{Pa}(x)]}{M_{\theta^t}[\mathrm{Pa}(x)]}$$

At each iteration of the algorithm, EM is guaranteed to increase the likelihood of the data given the model, and it has been proven that EM will converge to a local maximum in likelihood [12].

When learning Bayesian network structure with latent variables, a variant of EM called structural EM is often used [36]. As in the EM algorithm for parameter estimation, structural EM alternates between an expectation step and a maximization step. In the expectation step, the current model is used to generate a completed data set,

which is used to compute the expected sufficient statistics. In the maximization step, the expected sufficient statistics are used to improve the model. The primary difference is that the maximization step in structural EM will improve both the parameters and the structure. The structure learning algorithm used in the maximization step can be any of the search algorithms used in the complete data case. If the scoring metric used is based on the probability of the data given the model and the search algorithm used will improve the score at each iteration, then the structural EM procedure will converge to a local maximum. However, since EM is a local optimization method, it will often converge to sub-optimal parameter estimates.

2.6 Particle Swarm Optimization

The Particle Swarm Optimization (PSO) algorithm was first proposed by Eberhart and Kennedy [37]. 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 (p_i). These coordinates are called the personal best position of the particle. The algorithm also keeps track of the overall best solution value and location found so far by any particle in the population (p_g). This is called the global best position.

Both position and velocity are typically defined as a vector 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 locations of p_i and p_g relative to the current position. Eventually all

Algorithm 2.6 Particle Swarm Optimization

Procedure PSO(
ϕ_1 // maximum force with which a particle is pulled toward p_i;
ϕ_2 // maximum force with which a particle is pulled toward p_g;
ω // inertia weight
)

1: Initialize the particles
2: **repeat**
3: **for** each particle position $x_i \in \mathbf{P}$ **do**
4: Evaluate position fitness $f(x_i)$
5: **if** $f(x_i) > f(p_i)$ **then** $p_i = x_i$
6: **if** $f(x_i) > f(p_g)$ **then** $p_g = x_i$
7: $v_i = \omega v_i + \mathrm{U}(0, \phi_1) \otimes (p_i - x_i) + \mathrm{U}(0, \phi_2) \otimes (p_g - x_i)$
8: $x_i = x_i + v_i$
9: **end for**
10: **until** termination criterion is met

particles move closer to an optimum in the search space. The pseudocode for the traditional PSO algorithm is presented in Algorithm 2.6.

PSO begins by randomly initializing a swarm of particles over the search space. On each iteration of the algorithm, a particle's fitness is calculated using a problem dependent fitness function. The personal best position is maintained in the vector p_i. The global best position found among all particles is maintained in the vector p_g. At the end of each iteration a particle's velocity, v_i, is updated based on p_i and p_g. The use of both personal best and global best positions in the velocity equation ensures diverse responses within the swarm.

In Algorithm 2.6, $\mathbf{P}$ is the particle swarm, $\mathrm{U}(0, \phi_i)$ is a vector of random numbers uniformly distributed in $[0, \phi_i]$, $\otimes$ is component-wise multiplication, v_i is the current velocity of a particle, and x_i is the object parameters or current 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.

The random weighting of the parameters ϕ_1 and ϕ_2 in PSO can result in an explosion in which the particles' velocity and position rapidly move toward infinity. While many researches have addressed this problem with the use of a maximum velocity V_{max} and the above-mentioned inertia weight ω, Clerc and Kennedy [38] have suggested introducing a constriction factor χ to the velocity update equation as follows:

$$v_i = \chi(\omega v_i + \mathrm{U}(0, \phi_1) \otimes (p_i - x_i) + \mathrm{U}(0, \phi_2) \otimes (p_g - x_i))$$

The authors recommend setting the constriction factor χ relative to $\psi = \phi_1 + \phi_2$ as shown below:

$$\chi = \frac{2}{|2 - \psi - \sqrt{\psi^2 - 4\psi}|}$$

Note that the authors assume that $\psi = \phi_1 + \phi_2 > 4$. By setting the constriction factor as recommended above, the authors eliminate the need for a maximum velocity.

Often, the reliance on a global best for updating the velocity of each particle can cause PSO to get trapped in local optima. Additional variants that do not rely on the entire swarm's best known position when updating velocity have been developed to address this shortcoming [39, 40, 41, 42]. One commonly used alternative topology is the ring topology, in which each particle uses the position of two of its local neighbors to update its velocity [42].

2.7 Discrete Particle Swarm Optimization

Kennedy and Eberhart proposed a modification of the traditional PSO algorithm for problems with binary-valued solution elements [43]. In this algorithm, each particle's position is a vector from the d-dimensional binary solution space $x \in \{0,1\}^d$, and each particle's velocity is a vector from the d-dimensional continuous space, $v_i \in \mathbb{R}^d$.

A particle's velocity denotes the probability of that particle's position having a value of 0 or 1 in the next iteration. Each particle's velocity is updated as in traditional PSO, while each particles position is updated using the following equation:

$$p(x_i = 1) = \frac{1}{1 + \exp(-v_i)}$$

Although this algorithm was been shown to be effective, it is limited to problems with binary valued solution elements.

A discrete multi-valued PSO (DMVPSO) algorithm, which relaxed this binary state assumption, was proposed by Veeramachaneni *et al.* [44]. 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 and is updated as shown 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:

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

Notice that, while there is a probability of choosing any value between $[0, M-1]$ for a given S_i, the probability of choosing a given value decreases based on its distance from the current value of S_i.

2.8 Overlapping Swarm Intelligence

Overlapping Swarm Intelligence(OSI) is a multi-population search technique first proposed by Haberman and Sheppard to develop an energy-efficient routing protocol for sensor networks [15]. OSI is a variant of PSO in which a problem is divided into a series of overlapping sub-problems, and a swarm is assigned to each. Each swarm optimizes over its assigned sub-problem, and a competition mechanism is used to combine the solutions found by the various swarms. The specific sub-problem assigned to each swarm is often based on the underlying conditional dependencies of the problem [45, 46].

More formally, for some function $F(\cdot)$ to be optimized with parameters $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$, OSI assigns a subset of the parameters $\mathbf{Y}_i \subseteq \mathbf{X}$ to each swarm s_i. Each swarm in OSI optimizes over its subset of parameters via PSO, and the algorithm maintains a global solution vector for the complete set of parameters $\mathbf{x_g} = \{x_1, x_2, \ldots, x_n\}$ to be used for fitness evaluation.

The fitness of each particle is evaluated by inserting its partial solution into a complete global solution vector and evaluating the quality of the newly formed solution. For a particle $p_{i,j}$ in swarm s_i, a complete solution vector $\mathbf{z}_{i,j}$ is constructed

from the particle's partial solution vector $\mathbf{y}_{i,j}$ as follows:

$$\mathbf{z}_{i,j} = \mathbf{y}_{i,j} \cup \mathbf{x}_g \backslash \{x_k : X_k \in \mathbf{Y}_i\}$$

The fitness of this newly constructed complete solution vector $\mathbf{z}_{i,j}$ is then evaluated and used as the fitness for particle $p_{i,j}$.

When multiple swarms optimize over the same portion of the global solution vector, these swarms are said to overlap. At the end of each iteration, overlapping swarms compete to determine which values to include for each entry in the global solution vector. This competition is held between the values encoded in the best particles of each swarm. The parameters resulting in the highest fitness are selected for inclusion in the global solution vector. In some cases, swarms may be seeded with values from the global solution vector once the competition is complete.

CHAPTER 3

RELATED WORK

Here we present a review of literature related to distributed soft computing, distributed optimization, and multi-population algorithms. We also describe work done in the areas of inference, parameter estimation, and structure learning in Bayesian networks.

3.1 Distributed Soft Computing

Several authors have proposed distributed multi-population genetic algorithms (GA) [47, 48, 49, 50]. 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 [48] when applied to the problems of neural network parameter learning, the traveling salesman problem, and several deceptive problems proposed by Goldberg *et al.* [51]. One type of multi-population algorithm is the island model, in which subpopulations called islands are maintained and individuals in each subpopulation are allowed to migrate. 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 [50]. Like OSI, these methods maintain multiple subpopulations, but, unlike OSI, each of the sub-populations learn complete solutions to the problem being optimized.

Van den Bergh and Engelbrecht developed several distributed PSO methods for the training of multi-layer feed-forward neural networks [52]. These methods include NSPLIT in which there is a single particle swarm for each neuron in the network, LSPLIT in which there is a swarm assigned to each layer of the neural network, and ESPLIT which splits the serialized vector in half, assigning a swarm to each split. 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 sub-problems. This approach is called Overlapping Swarm Intelligence (OSI) [15, 53, 54]. 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. As distinct from NSPLIT, LSPLIT, and ESPLIT, here overlap occurs, requiring 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 [15]. 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. Thus, 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 [53]. Deep neural networks are defined as neural networks that have more than two hidden layers. 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 Fortier *et al.* [54].

3.2 Distributed Optimization

Much work has been done in the area of distributed optimization. Rabbat and Nowak [55] analyzed the convergence of distributed optimization algorithms in sensor networks. Many algorithms used to estimate environmental parameters or functions of interest in sensor networks transmit all data to a central point for processing. Rabbat and Nowak propose an alternative approach based on distributed in-network processing. The authors showed that, by distributing the computation of functions of interest in sensor networks, communication and energy resources consumed can be reduced significantly. The authors proved that, for a large set of problems, these algorithms will converge to a solution within a certain distance of the global optimum.

In a network of agents, consensus means that the agents reach an agreement regarding a certain value of interest that depends on the state of all agents. Problems that require agents in such a network to reach consensus are called consensus problems. A consensus algorithm is an interaction rule specifying the information exchange between an agent and its neighbors in the network. Olfati-Saber *et al.* [56] provided an analysis of consensus algorithms for multi-agent networked systems. The authors defined several types of consensus problems, described methods of convergence and performance analysis for multi-agent distributed systems, and proved the convergence of these methods.

Patterson *et al.* [57] provided an analysis of the convergence rate for the distributed average consensus algorithm. In the distributed average consensus algorithm, each node in the network has some initial value, and the goal is for all nodes to reach consensus on the average of the values for all nodes. Consensus must be reached using only communication between neighbors in the network graph. This work also included an analysis of the relationship between the convergence rate and the network topology.

Boyd *et al.* [58] analyzed convex distributed optimization in the context of machine learning and statistics. The authors argued that the alternating direction method of multipliers (ADMM) can be applied to such distributed optimization algorithms. In ADMM, a problem is divided into small local subproblems that are solved and used to find a solution to a large global problem. The authors showed that this approach can be applied to a wide variety of distributed optimization problems.

Distributed Overlapping Swarm Intelligence (DOSI) has several similarities to the methods discussed in this section. As in the first three works, DOSI requires consensus to be reached between all swarms in the algorithm with respect to the state assignment for a Bayesian network. Like the distributed average consensus algorithm

discussed by Patterson *et al.*, consensus between the swarms must be reached using only communication between neighbors in the network graph. However, unlike Patterson's distributed average consensus algorithm, consensus in DOSI is not reached with respect to an average value, but the swarms instead reach a consensus with respect to a value determined by a designated arbiter swarm. To determine the best value for a given variable, the arbiter swarm holds a competition between the values learned by all swarms that are optimizing over the variable. The winning value is then propagated to the other swarms to ensure consensus. The difference between this method and the average consensus algorithm proposed by Patterson is significant, because once consensus is reach in DOSI, not only will all swarms agree on a value for a particular variable, but this value will be always be the value that achieved the highest fitness competition. DOSI is similar to the work by Boyd *et al.* since it solves subproblems in a distributed manner and recombines the solutions to answer a larger global problem.

3.3 Common Approaches to Bayesian Inference

As mentioned in Section 2.2.3, an exact algorithm to solve the k Most Probable Explanation (k-MPE) problem called bucket elimination was proposed by Dechter [27]. This algorithm uses a form of variable elimination in which the node with the fewest neighbors is eliminated at each iteration. Bucket elimination uses max-marginalization instead of sum-marginalization when eliminating a variable and stores the most probable state assignment for the variable. In max-marginalization, the sum in equation 2.1 for sum-marginalization is replaced with a max operator. Like variable elimination, this algorithm has worst-case time complexity that is exponential in the

tree width of the network, where the tree width of a graph is the size of the largest maximal sub-clique minus one.

An approximation algorithm for the MPE problem called mini-bucket elimination (MBE) is described in [59]. While MBE can be applied to problems other than abductive inference, here we only examine it in the context of abductive inference. This algorithm is a variation of bucket elimination that can address both partial and full abductive inference. MBE further partitions the buckets defined in the bucket elimination algorithm into mini-buckets, and the elimination operator is applied to these mini-buckets instead of being applied to each singleton function in a bucket. The size of the mini-bucket partitions can be modified so that the algorithm produces exact solutions to the abductive inference problem. This is done by using the complete buckets of exact bucket elimination, instead of partitioning into mini-buckets.

A divide and conquer algorithm that provides an exact solution to the k-MPE problem is described by Nilsson in [60]. Nilsson's approach is based on the flow propagation algorithm proposed in [61] for finding the k-MPE for clique trees. While the algorithm is faster than other exact abductive inference algorithms such as bucket elimination, it still has exponential time complexity and is impractical for large networks. This limitation is true of all exact inference algorithms.

A simulated annealing algorithm (SA) for partial abductive inference was proposed by de Campos as an approximation algorithm [62]. In simulated annealing the system is initialized to some initial temperature T_0. Then some change e 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 $p > 1$, 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.

The simulated annealing algorithm proposed by de Campos uses an evaluation function based on clique tree propagation. The algorithm begins with a single state assignment and at each iteration a single variable is modified within the state assignment. A hash table is maintained consisting of ⟨assignment, probability⟩ pairs and each new assignment is stored in this table. If an assignment is found that is not stored in the hash table, then the probability of the assignment is computed. Since SA only modifies a single state at each iteration, the algorithm can avoid recalculating all of the initial clique potentials when evaluating a new state assignment.

A stochastic local search (SLS) algorithm for solving the MPE problem was proposed in [63]. In this approach, stochastic local search was combined with an approximate inference algorithm called Gibbs Sampling. The results of the author's experiments indicate that their approach outperforms other techniques such as stochastic simulation, simulated annealing, or hill-climbing alone.

The Elvira Consortium software environment uses a junction tree based algorithm to approximate a solution to the k-MPE Problem [64]. This algorithm is based on Nilsson's algorithm, but approximate probability trees are used in place of the true probability trees.

3.4 Soft Approaches to Bayesian Inference

Several researchers have used soft computing techniques to find approximate solutions to the k-MPE problem. Gelsema describes a genetic algorithm for full abductive inference (GA-Full) in Bayesian networks [65]. In this approach, the states of the variables in the Bayesian network are represented by a chromosome corresponding

to a vector of Boolean values. Each value in the chromosome corresponds to a state assignment for a node in the network. Crossover and mutation are applied to the chromosomes to generate offspring from parent chromosomes. To evaluate chromosome fitness, the chain rule is applied.

A graph-based evolutionary algorithm for performing approximate abductive inference on Bayesian Networks was developed by Rojas-Guzman *et al.* [66]. In their method, the graphs specify a possible solution that is a complete description of a state assignment for a Bayesian network. Fitness of a chromosome is based on the absolute probability of the chromosome's corresponding assignment. However, this approach was designed to find a single most probable explanation rather than the $k > 1$ most probable assignments.

A genetic algorithm for partial abductive inference was proposed by de Campos *et al.* [67]. In this approach, the state assignments for the subset of variables are represented as a chromosome consisting of integers. The value of each integer in a chromosome corresponds to the state of a variable in the network. To evaluate the fitness of each chromosome, clique tree propagation is used.

Sriwachirawat *et al.* [68] developed a niching genetic algorithm (NGA) to find k-MPE, designed to utilize the "multi-fractal characteristic and clustering property" of Bayesian networks. The multi-fractal characteristic and clustering property states that low-probability instantiation clusters and high-probability instantiation clusters are normally located far away from others, while, medium-probability instantiations do not form clusters and are located almost everywhere in the search space. This algorithm makes use of the observation that there are regions within the joint probability distribution of the Bayesian Network that are highly "self-similar." Because of this self-similarity, the authors chose to organize their GA using a probabilistic

crowding method that biases the crossover operator toward self-similar individuals in the population. Chromosomes in this approach were encoded as described in [65].

Ganesan Pillai and Sheppard [69] describe a discrete multi-valued PSO (DMVPSO) [44] approach for finding k-MPE. In their algorithm, each particle's set of object parameters is represented by a string of integers corresponding to state assignments for each node in the network. The chain rule is used to calculate the fitness of each particle. The results of the authors' experiments indicated they were able to find competitive explanations much more efficiently than the approaches described in [65] and [68].

3.5 Common Approaches to Bayesian Parameter Estimation

A number of algorithms have been developed for parameter estimation. As described in section 2.5, the most common algorithm for parameter estimation is Expectation Maximization (EM) [12]. The algorithm begins with an initially random parameter assignment and repeatedly executes the expectation and maximization steps. During the expectation step, the algorithm samples 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 variables 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 efficiency 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 [70]. 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 [71].

In 2002, Elidan *et al.* used data perturbation to improve upon the quality of the local maxima reached by EM [72]. 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 [73]. This approach, which is based on the information bottleneck framework described by Tishby *et al.* [74], involves grouping observed variables by mutual information and then creating a hidden variable for each group.

3.6 Soft Approaches to Bayesian Parameter Estimation

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[75].

Another variant of EM based on evolutionary computation was proposed by Mengshoel *et al.* in 2012 [76]. 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.

3.7 Common Approaches to Bayesian Structure Learning

Yuan *et al.* [77] propose an exact algorithm for Bayesian structure learning based on the A* algorithm which is capable of find an optimal Bayesian network structure. This is a state space search technique in which heuristics are used to search only among the most promising portions 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.

The Greedy Thick Thinning algorithm (GTT) described by Heckerman [78] 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 [78]. This algorithm starts with an initially empty graph and at each iteration an edge e is randomly is selected. A change is made to this edge with probability $p = \exp(\Delta(e)/T_0)$ as described in section 3.3. These changes can include removing, reversing, or adding an edge.

Cooper *et al.* describe a method for constructing Bayesian networks from data called K2 [28], which we describe in more detail in section 2.3. 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 $P(B, \mathbf{D})$ as the scoring function.

3.8 Soft Approaches to Bayesian Structure Learning

Several researchers have applied soft computing techniques to the problem of Bayesian structure learning. Wang *et al* proposed a PSO algorithm for Bayesian structure learning [9] where the position of each particle is an $n \times n$ adjacency matrix, and n is the number of variables in the network. The fitness function for a candidate network in this approach is denoted as:

$$F(B) = \log P(\mathbf{D}|B)$$

where $\mathbf{D}$ denotes the training data and B denotes the Bayesian network. 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* [10]. In this algorithm each chromosome consists of an ordering of the nodes in the network. For a given ordering, each node can only be 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 [28] to the ordering. The fitness function for a candidate network in this approach is denoted as:

$$F(B) = P(B, \mathbf{D}) \tag{3.1}$$

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 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 3.1. The order of the nodes in the graph traversal containing the highest pheromone levels defines the variable ordering for the K2 algorithm.

.

CHAPTER 4

FULL ABDUCTIVE INFERENCE IN BAYESIAN NETWORKS

In many domains, when reasoning with a Bayesian network, it is desirable to know the most probable explanation of the evidence, rather than the probability of a single variable in the system. The problem of performing such queries is called abductive inference. In full abductive inference, the goal is to find the most probable states for all variables in the network, given the evidence. Unfortunately, full abductive inference is known to be NP-Hard and Dagum [14] showed, that even the problem of finding a constant factor approximation for the abductive inference problem is NP-hard.

To demonstrate the validity of applying OSI to problems of full abductive inference we present both distributed and centralized OSI-based algorithms for full abductive inference, and we compared our algorithms to existing approaches.

4.1 Full Abductive Inference via OSI

We developed an approach for full abductive inference based on the OSI methodology [45]. In our approach, a swarm is associated with each node in the network. Each node's corresponding swarm learns the variable assignments associated with that node's Markov blanket using the DMVPSO algorithm [44]. This representation provides an advantage since every node in the network is conditionally independent of all other nodes when conditioned on its Markov blanket. The pseudocode for our approach is shown in Algorithm 4.1.

Algorithm 4.1 Overlapping Swarm Intelligence

Procedure OSI-Infer(
  G // A Bayesian network
  E // Evidence variables
)

Initialize global set of state assignments A using forward sampling
Initialize particles in each swarm
repeat
  for each swarm s **do**
    for each particle, $p \in s$ **do**
      Construct personal set of state assignments B_p
      Add B_p to A
      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
  $A \leftarrow k$ most probable assignments in A
  for each state assignment $\alpha \in A$ **do**
    for each node n in the network **do**
      Let S be all swarms s where $n \in \text{MB}(X_s)$
      Let v_n be the state of n in α
      $v_n \leftarrow Compete(S, n, \alpha)$
      $SeedSwarms(S, v_n)$
    end for
  end for
until termination criterion is met

Our algorithm requires that a set of k global state assignments A be maintained across all swarms for inter-swarm communication. A is initialized by forward sampling $m \geq k$ samples and selecting the k most probable assignments for inclusion in A. Forward sampling is a method used to generate a sample for every variable in a network, such that each sample is generated proportionally to its probability. Forward sampling draws a sample for each variable, following a topological ordering of the network. It samples a value for each variable from the distribution of the variable given the values assigned to its parents.

Each particle's position is defined by a d-dimensional vector of discrete values. Each position value corresponds to the state of a variable in the swarm's Markov blanket, thus each particle represents a state assignment for part of the network. We use log likelihood to determine the quality of a complete state assignment as follows:

$$\begin{aligned} L(\mathbf{x}) &= \log\left(\prod_{i=1}^{n} P(x_i|\mathrm{Pa}(x_i))\right) \\ &= \sum_{i=1}^{n} \log P(x_i|\mathrm{Pa}(x_i)). \end{aligned}$$

where $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ is a complete state assignment and $\mathrm{Pa}(x_i)$ corresponds to the assignments for the parents of x_i.

Given a partial state assignment $\mathbf{x}_p$ represented by some particle p in swarm s and the set of complete global state assignments $A = \{\alpha_1, \ldots, \alpha_k\}$, we can construct a new set of state assignments $B_p = \{\beta_1, \ldots, \beta_k\}$ for use in fitness evaluation by inserting x_p into each state assignment $\alpha_i \in A$ as follows:

$$\forall \beta_i \in B_p \quad \beta_i = \{x_p\} \cup \alpha_i \backslash \mathrm{MB}(X_s)$$

Algorithm 4.2 Compete

```
    Procedure Compete(
        S    // competing swarms
        n    // node the swarms are competing over
        α    // a state assignment
    )
 1: bestQ ← −∞
 2: for each swarm s ∈ S do
 3:     Let p_g be the most fit particle in s
 4:     Let v_n be the state of n within p_g
 5:     Insert v_n into α
 6:     if L(α) > bestQ then
 7:         bestQ ← L(α)
 8:         bestV ← v_n
 9:     end if
10: end for
11: return bestV
```

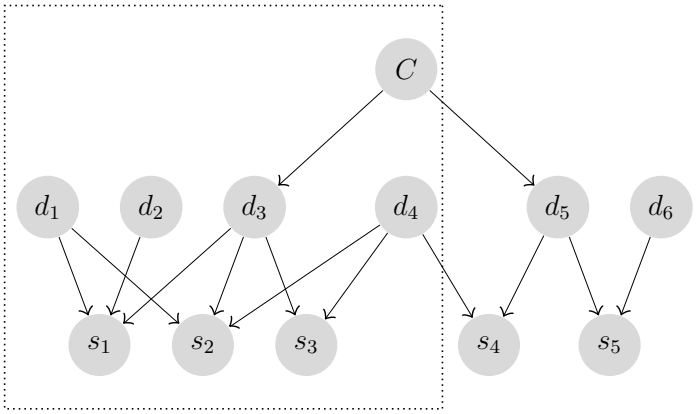
Algorithm 4.3 SeedSwarms

```
    Procedure SeedSwarms(
        S    // swarms to be seeded
        v_n  // state of node n
    )
1: for each swarm s ∈ S do
2:     Let p be the least fit particle in s
3:     Let q_n be the state of n in p
4:     q_n ← v_n
5: end for
```

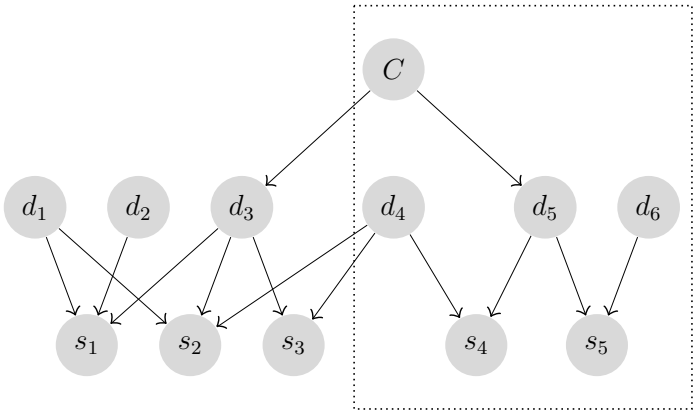
where $\text{MB}(X_s)$ consists of the state assignments for the Markov blanket of node X_s within A_i. We use B_p to calculate the fitness of each particle,

$$f(p) = \sum_{\beta_i \in B_p} L(\beta_i)$$

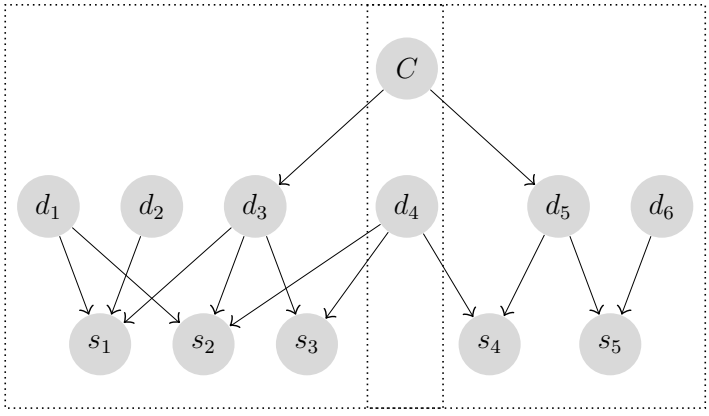
This function defines the fitness of particle p as the sum of the log likelihoods of the assignments in A when the value assignments encoded in p are substituted into A.



(a) Markov blanket of d_3



(b) Markov blanket of d_5



(c) Overlap of Markov blankets for d_3 and d_5

Figure 4.1: Markov blanket example

Our algorithm exploits the concept of Markov blanket overlap, which is illustrated in Figure 4.1. Figure 4.1(a) highlights the Markov blanket of d_3, Figure 4.1(b) highlights the Markov blanket of d_5, and Figure 4.1(c) highlights the overlap of 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 4.1(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.

When two or more swarms share a node in the network (such as c and d_4 in Figure 4.1(c)), these swarms are said to overlap. At the end of each iteration of the algorithm, overlapping swarms compete to determine which state is assigned to a given node in each assignment $\alpha_m \in A$ (Algorithm 4.2). This competition is held between the state assignments found by the personal best particles in each swarm. The state resulting in the highest log-likelihood is the one selected for inclusion. Once a node's state has been selected through competition, each swarm associated with the node is seeded with this state. This is performed by the *SeedSwarms* function. In our algorithms, we seeded a swarm by replacing the least fit particle in the swarm with the state v_n for node n, where v_n is the state of n within the most fit particle in the swarm. This allows for the transfer of information between different swarms that are trying to optimize over the same values. This process is shown in Algorithm 4.2. In the example presented in Figure 4.1, the swarms associated with d_3 and d_5 would compete to determine which state is assigned to the nodes c and d_4.

Whenever a state assignment is constructed, that assignment is stored in A. At each iteration, A is pruned so that it contains the k most probable assignments found so far. Once the algorithm has terminated, A is returned.

4.1.1 Computational Complexity

We first describe 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 pseudo code shown in Algorithm 4.1.

- **Fitness Evaluation**– First, we determine the complexity of evaluating the fitness of an individual particle. During fitness evaluation we compute the log likelihood for k candidate explanations, where the calculation of likelihood for a single explanation has a worst case complexity of $\mathcal{O}(n)$, where n is number of variables in the network. Thus, the fitness function complexity is $\mathcal{O}(kn)$.

- **Optimization Algorithm**– Next, we determine the complexity of the underlying PSO algorithm. We assume that the algorithm has $q = |P|$ individuals, where P is the set of all the individuals in the population. During each iteration, an individual, with n variables in the worst case, 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 $\mathcal{O}(kn)$. This is done q times, once for for each individual; therefore, assuming the algorithm performs e iterations, the total complexity is $\mathcal{O}(kneq)$.

  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 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 $\mathcal{O}(kneq)$. Since this is done $s = |\boldsymbol{S}|$ times, the total complexity of this step is $\mathcal{O}(skneq)$.

- **Competition**– In OSI, the competition step 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 competing individuals. Note that there are n variables. In the worst case, all swarms will overlap over every variable, resulting in n competing subpopulations, while each fitness evaluation has a complexity of $\mathcal{O}(nk)$. Therefore, the total complexity of the competition step in OSI is $\mathcal{O}(nn^2k) = \mathcal{O}(n^3k)$.

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

$$\begin{aligned}\mathcal{O}(\text{OSI}) &= \mathcal{O}(m(nkeqs + n^3k)) \\ &= \mathcal{O}(mnkeqs + mn^3k)\end{aligned}$$

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

$$\mathcal{O}(\text{OSI}) = \mathcal{O}(mn^2keq + mn^3k)$$

Based on these results, we can see that OSI will almost always be more computationally complex due to the n^3 factor caused by the competition phase. However, in practice, the worst case $\mathcal{O}(n^3k)$ complexity of the competition phase will be rare, and will only occur when all variables in the network share a Markov blanket.

4.2 Full Abductive Inference via DOSI

We also modified the OSI approach proposed in [45] to eliminate the need for a set of global state assignments to be maintained across all swarms. In the distributed approach, each swarm s maintains a set of personal state assignments denoted as

A_s. The most probable assignments learned by each swarm are communicated to the other swarms and inserted into the swarm's personal state assignment through a periodic communication mechanism. At each iteration of the algorithm, the swarm associated with a given node holds a competition between all swarms that share this node to determine the most probable state assignment. This state assignment then is communicated to the other swarms. Because this approach does not require a global network to be shared between swarms, the learning process can be distributed. Initially a set A is obtained by forward sampling $m \geq k$ samples. Each A_s is initialized as the k most probable assignments in A.

The fitness calculation and the calculation of a new set of state assignments for a given swarm s and particle p, denoted $B_{s,p}$, is similar to the calculation of B_p used in OSI. Given a partial state assignment x_p represented by some particle p in swarm s and the set of personal state assignments $A_s = \{\alpha_1, \ldots, \alpha_k\}$, we can construct a new set of state assignments $B_{s,p} = \{\beta_1, \ldots, \beta_k\}$ by inserting x_p into each state assignment $\alpha_i \in A_s$ as follows:

$$\forall \beta_i \in B_{s,p} \quad \beta_i = \{x_p\} \cup \alpha_i \backslash \mathrm{MB}(X_s)$$

where $\mathrm{MB}(X_s)$ consists of the state assignments for the Markov blanket of node X_s within α_i. We use $B_{s,p}$ to calculate the fitness of each particle,

$$f(p) = \sum_{\beta_i \in B_{s,p}} L(\beta_i)$$

This function defines the fitness of particle p in a swarm s as the sum of the log likelihoods of the assignments in α_s when the state assignments encoded in p are substituted into α_s. Algorithm 4.4 shows the pseudocode for DOSI.

As with OSI, a competition is held between overlapping the swarms at the end of each iteration of the algorithm. For each node, its associated swarm holds a competition between the states learned by all swarms that optimize over the node. We call this swarm the arbiter swarm. During this competition, the state resulting in the highest log likelihood, is selected for inclusion in the arbiter swarm's set of personal state assignments. After performing competition, the state values selected by the arbiter swarm are propagated to the other swarms in the algorithm. This competition and sharing procedure is shown in Algorithm 4.5.

Unlike OSI, once a node's state has been selected by the competition function, only the swarm associated with the node is seeded with the new state. This is done to reduce communication overhead. State assignments are shared using the ShareStates function described by Algorithm 4.6. To share state assignments, each swarm keeps track of a variable δ_n for each node n. These variables indicate the minimum distance between the swarm's node and n in the moralized graph of the Bayesian network (the graph obtained by connecting all unconnected nodes that have a common child).

For example, a value of zero for δ_n indicates that n is the swarm's corresponding node while a value of one for δ_n indicates that the swarm's node is in the Markov blanket of n. Values for δ_n are set initially to infinity for all nodes other than the node assigned to that swarm. Values for δ_n are set to 0 for the node assigned to the swarm. At each iteration of the communication the tentative distances between each pair of nodes are updated. A swarm s_j will communicate its value for a node n to another swarm s_i if $s_i.\delta_n > s_j.\delta_n$. This is done because nodes that have a smaller value for δ_n are closer to n. At each iteration, a swarm s will communicate, with their neighbors, the state values learned by s, and the state values received by other swarms that have communicated with s in the past. Thus, if a swarm s_i is a neighbor of swarm s_j, then s_i will receive the learned states from s_j in a single iteration, but if

s_i is a distance of two from s_j, then it will receive the learned states from s_j after two iterations. So, swarms whose nodes are closer to the node n will have a more current value for the state of n. This communication mechanism ensures that DOSI will reach consensus with respect to the values for δ_n and the assigned values for a given state. This means that, if the communication mechanism runs for enough iterations, then all swarms will agree on the values for δ_n and the assigned states for every node n.

A proof of DOSI consensus for connected nodes follows. This proof ties the number of iterations required for consensus between two nodes to the distance between the nodes in the moralized graph of the Bayesian network. Note that, in the moralized graph G of some Bayesian network B, an edge is added between all unconnected parents that share a child. As a result, all nodes in the moralized graph will have an edge connecting them to their children's parents. This means that, for each node X in the moralized graph, an edge will be present between X and every node in MB(X). As a result, if ShareStates(X, Y) is called from Algorithm 4.5, then an edge must exist between nodes X and Y in the moralized graph of the Bayesian network. Therefore, if $d(X, Y) = 1$, then $X \in \text{MB}(Y)$ and during the each iteration, ShareStates(Y, X) will be called. This concept if formalized in Lemma 1.

Algorithm 4.4 Distributed Overlapping Swarm Intelligence

```
    Procedure DOSI-Infer(
        G    // A Bayesian network
        E    // Evidence variables
        X    // Set of all variables in the network
    )
 1: Sample A using forward sampling
 2: Initialize particles in each swarm
 3: for each swarm s do
 4:     Create an empty list of assignments A_s
 5:     Add k most probable assignments in A to A_s
 6: end for
 7: repeat
 8:     for each swarm s do
 9:         for each particle, p ∈ s do
10:             Construct B_p
11:             Add B_p to A_s
12:             Calculate particle fitness f(p)
13:             if f(p) > p's personal best fitness then
14:                 Update p's personal best position and fitness
15:             end if
16:             if f(p) > the global best fitness then
17:                 Update global best position and fitness for s
18:             end if
19:             Update p's velocity and position
20:         end for
21:         A_s ← k most probable assignments in A_s
22:     end for
23:     CompeteAndShare(X)
24: until termination criterion is met
```

Algorithm 4.5 Compete and Share

```
    Procedure CompeteAndShare(
        X    // set of nodes
    )
 1:     for each node n ∈ X do
 2:         Let s be the swarm associated with n
 3:         for each state assignment α ∈ A_s do
 4:             Let S be all swarms s where n ∈ MB(X_s)
 5:             Let v_n be the state of n in A
 6:             v_n ← Compete(S, n, α)
 7:             SeedSwarm(s, v_n)
 8:         end for
 9:
10:         for i = 0 to C do
11:             for each swarm s_i do
12:                 for each swarm s_j ∈ MB(X_{s_i}) do
13:                     ShareStates(s_i, s_j)
14:                 end for
15:             end for
16:         end for
17:     end for
```

Algorithm 4.6 Share States

```
    Procedure ShareStates(
        s_i    // First swarm
        s_j    // Second swarm
    )
 1: Let A_i and A_j be the set of personal state assignments of s_i and s_j respectively
 2: for each node n do
 3:         if s_i.δ_n > s_j.δ_n then
 4:             s_i.δ_n ← s_j.δ_n + 1
 5:             for m = 0 to k do
 6:                 Let α_i be the m^th assignment in A_{s_i}
 7:                 Let α_j be the m^th assignment in A_{s_j}
 8:                 Insert α_j's value for n into α_i
 9:             end for
10:         end if
11: end for
```

Lemma 1. *[Moralization and Markov Blankets] Let X and Y be two nodes in some Bayesian network $B = (\mathbf{V}_B, \mathbf{E}_B)$, and let $G = (\mathbf{V}_G, \mathbf{E}_G)$ be the moralized graph of the Bayesian network B, where $\mathbf{V}_G$ is the set of nodes in G and $\mathbf{E}_G$ is the set of edges. We prove by contradiction that*

$$(X, Y) \in \mathbf{E}_G \rightarrow X \in MB(Y) \tag{4.1}$$

Proof. Assume that that 4.1 is false. Then there exists an edge $(X, Y) \in \mathbf{E}_G$ such that $X \notin \text{MB}(Y)$. If $(X, Y) \in \mathbf{E}_G$, then either $(X, Y) \in \mathbf{E}_B$ or the edge (X, Y) was added to $\mathbf{E}_G$ during moralization.

If $(X, Y) \in \mathbf{E}_B$, then either $Y \in \text{Pa}(X)$ or $X \in \text{Pa}(Y)$. This implies, by the definition of the Markov blanket, that $X \in \text{MB}(Y)$.

Alternativly, if the edge (X, Y) was added to $\mathbf{E}_G$ during moralization then, by the definition of moralization, there exists some node Z such that $X \in \text{Pa}(Z)$ and $Y \in \text{Pa}(Z)$. This implies that X is Y's children's parent. Therefore, by the definition of the Markov blanket, $X \in \text{MB}(Y)$.

Thus, either $(X, Y) \in \mathbf{E}_B$, implying that $X \in \text{MB}(Y)$, or the edge (X, Y) was added to $\mathbf{E}_G$ during moralization, also implying that $X \in \text{MB}(Y)$. Therefore $X \in \text{MB}(Y)$ and $X \notin \text{MB}(Y)$, a contradiction.

Conclusion: By the principle of contradiction, 4.2 is true. □

Theorem 1 (DOSI Consensus)**.** *Let X and Y be two nodes in some Bayesian network $B = (\mathbf{V}_B, \mathbf{E}_B)$ where $\mathbf{V}_B$ is the set of nodes in B and $\mathbf{E}_B$ is the set of edges. Let $C(X, Y)$ be the number of sharing iterations required (the loop on line 10 of Algorithm 4.5) for X and Y to have reached consensus with respect to the state of node X. We prove by induction that*

$$C(X, Y) = d(X, Y) \tag{4.2}$$

where $d(X, Y)$ is the minimum distance between X and Y in the moralized graph $G = (\mathbf{V}_G, \mathbf{E}_G)$ of the Bayesian network B.

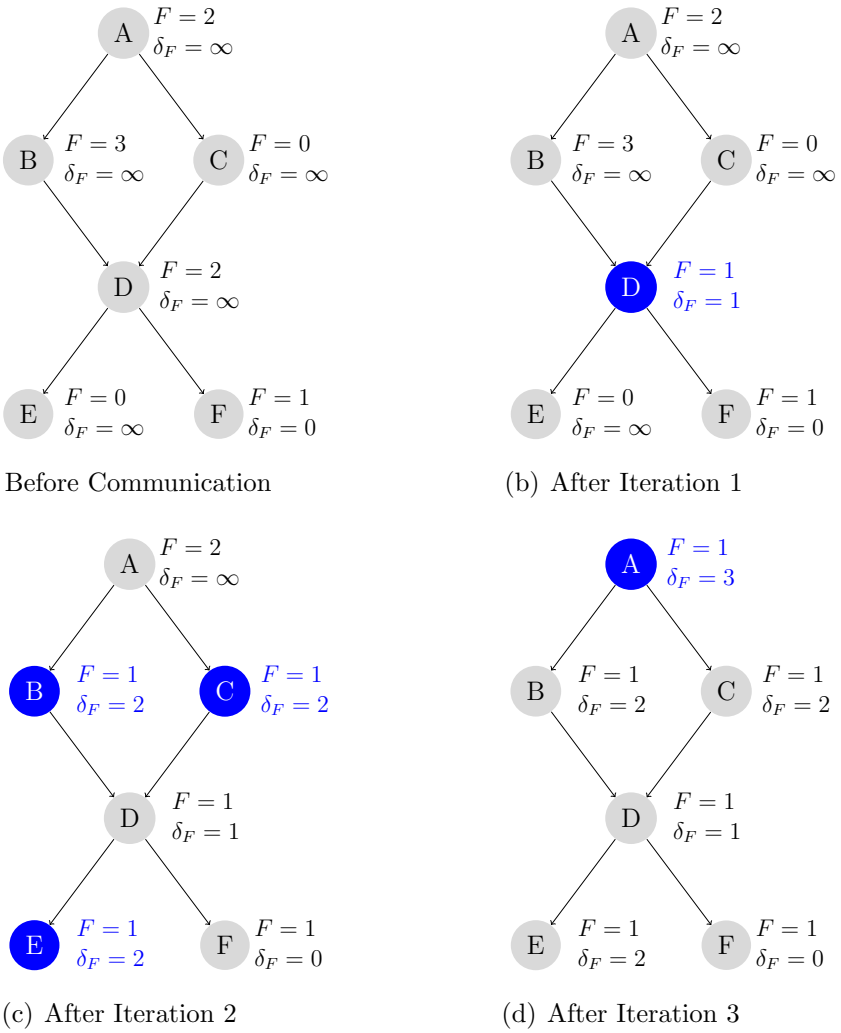
Proof. Base case: Let i be the number of elapsed sharing iterations. Suppose $d(X, Y) = 1$. When $i = 0$, $X.\delta_X = 0$ and $Y.\delta_X = \infty$. Since $d(X, Y) = 1$, we know that $(X, Y) \in \mathbf{E}_G$. By Lemma 1, this implies that $X \in \text{MB}(Y)$. Thus, during the first iteration, ShareStates(Y, X) will be called. Because $Y.\delta_X > X.\delta_X$ when $i = 0$, $Y.\delta_X \leftarrow X.\delta_X + 1 = 0 + 1 = 1$. Also, the state assignments for node X in A_Y will be identical to the state assignments for node X in A_X. Therefore consensus will be reached when $i = 1$ in this case. Thus $C(X, Y) = 1 = d(X, Y)$.

Induction step: The inductive hypothesis is that $C(X, Y) = d(X, Y)$ holds for $d = k$. Suppose $d(X, Z) = k + 1$. Then there must exist some node Y such that $(Y, Z) \in \mathbf{E}_G$ and a path with length $d(X, Z) = k + 1$ which passes through Y. Therefore, there must exist a path between X and Y such that $d(X, Y) = k$. Based on the inductive hypothesis, once $i = k$ consensus will be reached between X and Y with respect to the state of node X.

When $i = k + 1$, $Y.\delta_X = k$ and $Z.\delta_X = \infty$. By Lemma 1, we know that, since $(Y, Z) \in \mathbf{E}_G$, the MB of Y must include Z. Thus, when $i = k + 1$, ShareStates(Z, Y) will be called. Because $Z.\delta_X > Y.\delta_X$, $Z.\delta_X \leftarrow Y.\delta_X + 1 = k + 1$. Additionally, the state assignments for node X in A_Z will become identical to the state assignments for node X in A_X. Therefore consensus will be reached after $i = k + 1$ in this case. Thus $C(X, Y) = k + 1 = d(X, Y)$.

Conclusion: By the principle of induction, 4.2 is true for all $d(X, Y) \in \mathbb{N}$. □

An example of several iterations of the communication portion of the algorithm is shown in Figure 4.2. Here the most probable state assignment for F in each personal state assignment is shown along with each swarm's value for δ_F at the beginning of



(a) Before Communication

(b) After Iteration 1

(c) After Iteration 2

(d) After Iteration 3

Figure 4.2: Share States example

each iteration. In this example the swarm associated with node F has determined through competition that the best state assignment for F is 1. Prior to the first iteration of communication $\delta_F = 0$ for the swarm associated with F while $\delta_F = \infty$ in all other swarms. After the first iteration

$$D.\delta_F = F.\delta_F + 1 = 1$$

since $D.\delta_F = \infty > 0 = F.\delta_F$. When D updates its value for δ_F, it receives the most probable state assignment, $F = 1$, from the swarm associated with F. During the second iteration of communication D will share states with nodes B, C, and E. Once the second iteration is complete nodes B, C, and E have all set $F = 1$ in their personal state assignment and B, C, and E have all set

$$\delta_F = D.\delta_F + 1 = 2$$

During the third iteration, node B will share states with node A. Since $A.\delta_F = \infty > 2 = B.\delta_F$ the swarm associated with A will update its value for δ_F such that

$$A.\delta_F = B.\delta_F + 1 = 3$$

and A will set $F = 1$ based on the state of F in the personal state assignment for node B. Once the third iteration is complete, for any given node in the network, δ_F will contain the minimum distance between that node and F in the moralized graph of the network and all nodes will agree upon a value for F. Once the personal state assignment for all swarms have identical state assignments for some variable, we say that the network has reached consensus with respect to that variable. In this example, after the third iteration of communication the swarms will reach consensus with respect to F. Eventually all tentative distances will be updated in this way so that they reflect the correct distances between the nodes.

This algorithm is similar to Dijkstra's algorithm in that initially each node is assigned a tentative distance value for each δ_n (zero for our initial node and infinity for all other nodes) that is updated at each iteration through the communication mechanism. After each iteration, each node will update one of its δ's if and only if

one of its neighbors has a lower δ value for the same node. Because each node starts with a δ value of 0 for itself and ∞ for all others, the only δ's that will be updated are nodes that are within each other's MBs; therefore, after the first iteration, all δ's will either be 0, 1, or ∞.

As this process is repeated, the δ will be increased by one for each node as the value is distributed throughout the network, which also corresponds to the distance from one node to another node in the moralized graph. Eventually, all of the nodes will have finite numbers for all δ's in a connected network. If some parts of the network are disjoint from others, then there will still exist some delta's with values of ∞ for certain nodes. However, this does not present a significant problem. If two sub-networks are disjoint from one another, then the most probable state assignments for variables in one sub-network will be independent of the most probable state assignments in the other. This means that the most probable state assignments in one sub-network can be optimized without regard for the states assigned to variables in the other, so long as the two sub-networks are disjoint.

4.3 Experimental Design

For our first set of experiments, we compare our OSI and DOSI algorithms for full abductive inference to the following algorithms:

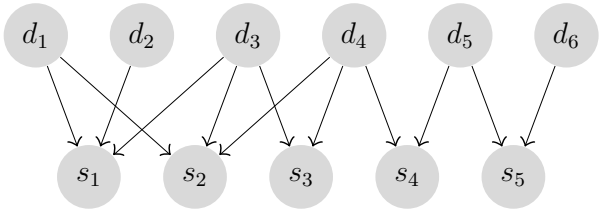
- SLS: Stochastic Local Search algorithm proposed in [63].
- NGA: Niching genetic algorithm proposed in [68].
- MBE: Mini bucket elimination algorithm proposed in [59].
- DMVPSO: Discrete multi-valued PSO algorithm proposed in [69].

Table 4.1: Properties of the test networks

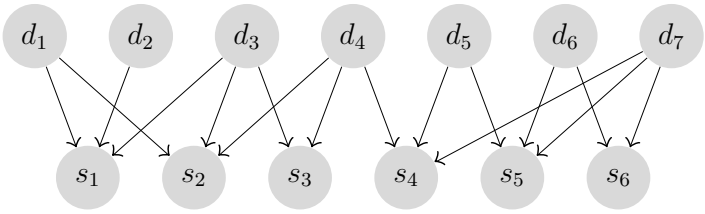
Network	Nodes	Arcs	Parameters	Ave. MB Size	Ave. States
Network A	11	12	261	4	3.00
Network B	13	16	399	4.53	3.00
Network C	15	12	483	4.60	3.00
Win95pts	76	112	574	5.92	2.00
Insurance	27	52	984	5.19	3.30
Hailfinder	70	66	2656	3.54	3.98
Hepar2	56	1236	1453	3.51	2.31

For these comparisons, we used the bipartite networks presented in Figure 4.3 [69] along with four additional Bayesian networks obtained from the Bayesian Network Repository [79]: Win95pts, Insurance, Hailfinder, and Hepar2. Win95pts and Hepar2 were chosen because they were the only large networks (60 - 100 nodes) in the bnlearn repository. Such large networks were desirable to validate our hypothesis that OSI and DOSI would excel when learning state assignments for larger networks. Insurance and Hailfinder were chosen because they are real-world networks that have been validated in published literature [80, 81]. The properties for all networks are shown in Table 4.1. For all networks each leaf node in the network was chosen as evidence with a 50% probability. The state of each evidence variable was chosen uniformly at random.

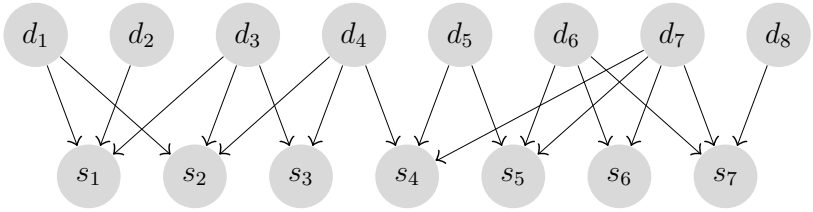
For each network, experiments were performed with different values of k: $k = 2, 4, 6, 8$. We did not examine the performance of OSI when $k = 1$ because the performance of our algorithms in this case will likely be similar to the performance when optimizing over multiple explanations. For all of the algorithms, initial populations were generated using forward sampling. In every experiment, the number of particles in each swarm was set to 20 and σ was set to 0.2. We chose a population size of 20 because we found that this resulted in good performance during preliminary evaluation of the algorithms. For the swarm-based algorithms ϕ_1 and ϕ_2 were set to



(a) Network A



(b) Network B



(c) Network C

Figure 4.3: Bipartite Bayesian networks used for experiments.

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 [82]. The value for σ was taken from [69] to ensure consistency of results. For the genetic algorithms, the population size was set to 20 to match the number of particles in each swarm. All algorithms were run until convergence. We determined that an algorithm had converged if the best solution found by the algorithm did not improve for 10 iterations. The sums of the log likelihoods for the k most fit solutions found in each run were averaged over ten runs of each algorithm. We compared the results using a paired t-test with a confidence interval of 95% to evaluate significance. In addition to log likelihood,

we also measured the number of fitness evaluations required by each algorithm for a comparison of computational complexity.

For networks A, B, and C, the MBE algorithm is used to compute the exact solution for the full abductive inference problem. For all other networks, MBE is used to find an approximate solution with parameters m and i set to 2 and 3, respectively, since the networks are too large for exact inference.

For our second set of experiments we performed a lesion study [83] by implementing two alternative versions of DOSI. In the first implementation (DOSI-Comm), the competition mechanism was disabled, while in the second implementation (DOSI-Comp), the communication mechanism was disabled. We compared these alternative implementations to DOSI to validate our hypothesis that both the competition and communication of DOSI improve the algorithm's performance in terms of average log likelihood of solutions found. All other parameters and design decisions were the same as the first set of experiments.

For the final set of experiments we compared the performance of various swarm architectures. In the first architecture (Markov), a swarm learns the state assignments for the Markov blanket of its corresponding node. In the second architecture (Clique), a swarm was assigned to each clique in the network's corresponding clique tree and each swarm learned the state assignments for all nodes in its associated clique. In the third architecture (Parent), a swarm was assigned to each node and each swarm learned the state assignments for the parents of its corresponding node. For the final architecture (Random), a swarm was assigned to each node and each swarm learned the state assignments for a randomly selected set of other nodes. For these experiments, we set $k = 1$, no evidence was set, and the log-likelihood of each architecture was averaged over 30 runs.

We have four hypotheses relating to the experiments presented here. First, we hypothesize that OSI and DOSI will outperform the competing approaches for most experiments. Second, we hypothesize that OSI and DOSI tie statistically for best on most of the network presented here. Third, we hypothesize that removing the competition or communication mechanism from DOSI will cause a drop in performance for some of the experiments. Finally, we hypothesize that the Markov blanket overlap architecture will provide the best performance for most networks, and will never be outperformed by the other architectures.

4.4 Comparison Against Existing Algorithms

Tables 4.2, 4.3, 4.4, and 4.5 show the average sum of the log likelihoods for each algorithm and each value of k. Bold values indicate that the corresponding algorithm's performance is statistically significantly better than all other algorithms for the network given the corresponding value for k. Algorithms that tie statistically for best are bolded. Table 4.2 shows the average sum of the log likelihoods for the population based algorithms while Table 4.4 shows the average sum of the log likelihoods for MBE and SLS. Table 4.3 compares the average sum of the log likelihoods for OSI and DOSI.

For all networks containing more than 15 nodes we observe that, based on the paired t-tests on log likelihood, the OSI algorithm outperforms all other approximate algorithms, and DOSI is never outperformed by any approximate algorithms other than OSI. For Network A, all population based algorithms tie statistically when k is set to 2, OSI, DOSI, and DMVPSO tie statistically when k is set to 2 and 6, and OSI outperforms all other approximate algorithms when k is set to 8. For Network B, OSI, DOSI, and DMVPSO tie statistically when k is set equal to 4 while OSI and

Table 4.2: Comparison against population based approaches

Network	k	OSI	NGA	GA-Full	DMVPSO
Network A	2	**-13.64 ± 0.02**	**-14.06 ± 0.51**	**-14.16 ± 0.87**	**-13.95 ± 0.73**
	4	**-27.31 ± 0.10**	-29.40 ± 1.59	-29.42 ± 1.69	**-28.33 ± 2.29**
	6	**-40.92 ± 0.03**	-46.70 ± 2.77	-47.07 ± 3.77	**-43.07 ± 2.67**
	8	**-54.56 ± 0.03**	-60.43 ± 2.12	-64.52 ± 4.18	-59.03 ± 3.68
Network B	2	**-17.12 ± 0.07**	-18.11 ± 0.48	-18.39 ± 0.56	-18.23 ± 0.40
	4	**-34.62 ± 0.14**	-36.90 ± 1.14	-37.32 ± 1.49	**-36.66 ± 0.57**
	6	**-51.87 ± 0.24**	-57.18 ± 2.63	-58.63 ± 2.16	-57.40 ± 1.90
	8	**-69.26 ± 0.32**	-77.25 ± 1.69	-77.74 ± 2.84	-76.05 ± 2.02
Network C	2	**-16.05 ± 0.02**	**-17.23 ± 1.53**	**-17.61 ± 1.47**	**-17.08 ± 1.26**
	4	**-32.09 ± 0.05**	**-37.69 ± 2.87**	**-36.82 ± 1.73**	**-35.05 ± 1.52**
	6	**-48.18 ± 0.11**	**-57.09 ± 2.56**	**-57.46 ± 1.69**	**-55.77 ± 2.45**
	8	**-64.28 ± 0.19**	-78.68 ± 5.01	-77.51 ± 3.21	-74.67 ± 2.77
Win95pts	2	**-32.27 ± 5.59**	-2442.06 ± 545.94	-2866.92 ± 755.73	-941.05 ± 1236.95
	4	**-57.10 ± 3.85**	-4561.73 ± 1602.60	-5615.89 ± 2551.70	-2162.27 ± 2810.62
	6	**-90.21 ± 13.53**	-9580.33 ± 1727.51	-9314.64 ± 2208.09	-3858.00 ± 3559.27
	8	**-124.78 ± 20.97**	-13885.76 ± 1843.64	-13240.43 ± 4066.10	-5947.98 ± 5681.22
Insurance	2	**-24.63 ± 2.29**	-36.57 ± 6.80	-34.55 ± 2.99	-31.85 ± 2.38
	4	**-48.85 ± 3.04**	-76.92 ± 19.59	-73.66 ± 7.87	-79.56 ± 8.43
	6	**-74.02 ± 9.14**	-120.14 ± 13.54	-133.65 ± 26.96	-132.08 ± 15.91
	8	**-100.25 ± 7.25**	-196.69 ± 35.05	-179.64 ± 25.23	-170.20 ± 24.44
Hailfinder	2	**-69.51 ± 2.19**	-176.20 ± 234.93	-102.08 ± 5.72	-247.79 ± 313.59
	4	**-142.31 ± 3.85**	-355.61 ± 314.29	-502.07 ± 931.43	-425.46 ± 500.03
	6	**-210.29 ± 6.16**	-1210.99 ± 1091.34	-1069.50 ± 983.59	-1795.57 ± 1210.33
	8	**-278.88 ± 9.34**	-2217.57 ± 1809.46	-2282.79 ± 1061.93	-3986.79 ± 983.68
Hepar2	2	**-66.54 ± 0.00**	-79.40 ± 2.10	-80.79 ± 2.82	-72.92 ± 2.07
	4	**-134.19 ± 1.85**	-164.57 ± 3.87	-161.64 ± 4.02	-146.78 ± 7.59
	6	**-199.84 ± 0.67**	-249.92 ± 7.44	-253.03 ± 7.61	-217.70 ± 10.71
	8	**-405.12 ± 3.75**	-472.44 ± 5.72	-471.40 ± 7.42	-440.27 ± 11.95

DOSI tie statistically for best when k is set equal to 2. OSI outperforms all other approximate algorithms when k is set to 6 and 8. For Network C, all population based algorithms tie statistically for best when k is set equal to 2, 4, and 6. OSI outperforms all other approximate algorithms when k is set to 8. For networks A, B, and C, the average sum of the log likelihoods for OSI differs from the exact solution by at most 0.4 while, for DOSI, the average sum of the log likelihoods differs from the exact solution by at most 3.11. For Win95pts, OSI and DOSI tie statistically for best when k is set equal to 2, 4, and 6. For Insurance, OSI and DOSI tie statistically for

Table 4.3: Comparison of OSI and DOSI

Network	k	OSI	DOSI
Network A	2	**-13.64 ± 0.02**	**-13.85 ± 0.58**
	4	**-27.31 ± 0.10**	**-27.81 ± 0.70**
	6	**-40.92 ± 0.03**	**-41.82 ± 2.51**
	8	**-54.56 ± 0.03**	-55.27 ± 0.45
Network B	2	**-17.12 ± 0.07**	**-17.20 ± 0.16**
	4	**-34.62 ± 0.14**	**-35.59 ± 1.45**
	6	**-51.87 ± 0.24**	-52.66 ± 0.64
	8	**-69.26 ± 0.32**	-71.97 ± 2.19
Network C	2	**-16.05 ± 0.02**	**-16.71 ± 0.96**
	4	**-32.09 ± 0.05**	**-34.58 ± 4.24**
	6	**-48.18 ± 0.11**	**-50.74 ± 2.15**
	8	**-64.28 ± 0.19**	-67.08 ± 2.59
Win95pts	2	**-32.27 ± 5.59**	**-40.60 ± 10.96**
	4	**-57.10 ± 3.85**	**-125.07 ± 98.46**
	6	**-90.21 ± 13.53**	**-350.87 ± 368.13**
	8	**-124.78 ± 20.97**	-520.60 ± 517.70
Insurance	2	**-24.63 ± 2.29**	**-26.77 ± 4.14**
	4	**-48.85 ± 3.04**	**-54.05 ± 6.08**
	6	**-74.02 ± 9.14**	**-83.40 ± 9.83**
	8	**-100.25 ± 7.25**	**-102.37 ± 12.98**
Hailfinder	2	**-69.51 ± 2.19**	-86.74 ± 8.72
	4	**-142.31 ± 3.85**	-183.56 ± 16.64
	6	**-210.29 ± 6.16**	-260.12 ± 20.88
	8	**-278.88 ± 9.34**	-357.30 ± 30.19
Hepar2	2	**-66.54 ± 0.00**	**-67.28 ± 1.44**
	4	**-134.19 ± 1.85**	**-136.34 ± 2.64**
	6	**-199.84 ± 0.67**	-205.37 ± 5.68
	8	**-405.12 ± 3.75**	**-409.87 ± 6.13**

best for all a values of k. For Hepar2, OSI and DOSI tie statistically for best when k is set equal to 2, 4, and 8.

Table 4.4: Comparison against MBE and SLS

Network	k	OSI	SLS	MBE
Network A	2	**-13.64 ± 0.02**	-19.40 ± 2.58	**-13.64**
	4	**-27.31 ± 0.10**	-38.40 ± 4.12	**-27.27**
	6	**-40.92 ± 0.03**	-59.20 ± 3.69	**-40.91**
	8	**-54.56 ± 0.03**	-82.01 ± 8.47	**-54.55**
Network B	2	-17.12 ± 0.07	-24.65 ± 5.47	**-16.99**
	4	-34.62 ± 0.14	-48.90 ± 4.48	**-34.37**
	6	-51.87 ± 0.24	-72.73 ± 8.01	**-51.59**
	8	-69.26 ± 0.32	-93.71 ± 5.15	**-68.86**
Network C	2	**-16.05 ± 0.02**	-23.67 ± 4.68	**-16.03**
	4	**-32.09 ± 0.05**	-50.11 ± 5.89	**-32.07**
	6	**-48.18 ± 0.11**	-71.88 ± 6.51	**-48.10**
	8	64.28 ± 0.19	-98.24 ± 8.31	**-64.14**
Win95pts	2	**-32.27 ± 5.59**	-3529.16 ± 1215.55	-2992.69
	4	**-57.10 ± 3.85**	-8469.61 ± 2217.93	-5991.64
	6	**-90.21 ± 13.53**	-11412.51 ± 1872.39	-8996.11
	8	**-124.78 ± 20.97**	-14708.08 ± 1830.15	-12006.28
Insurance	2	**-24.63 ± 2.29**	-1004.74 ± 992.53	-39.76
	4	**-48.85 ± 3.04**	-2097.83 ± 1558.87	-83.85
	6	**-74.02 ± 9.14**	-3778.13 ± 1181.34	-125.14
	8	**-100.25 ± 7.25**	-4797.66 ± 1964.59	-170.11
Hailfinder	2	**-69.51 ± 2.19**	-2121.23 ± 1648.51	-90.44
	4	**-142.31 ± 3.85**	-6175.14± 2845.54	-186.99
	6	**-210.29 ± 6.16**	-9255.36 ± 3385.27	-274.01
	8	**-278.88 ± 9.34**	-10032.40 ± 1965.23	-368.84
Hepar2	2	**-66.54 ± 0.00**	-88.42 ± 9.07	-72.21
	4	**-134.19 ± 1.85**	-177.58 ± 9.46	-148.68
	6	**-199.84 ± 0.67**	-266.16 ± 9.42	-233.23
	8	**-405.12 ± 3.75**	-460.57 ± 25.72	-472.46

Table 4.5: Comparison against modifications of DOSI

Network	k	DOSI	DOSI-Comp	DOSI-Comm
Network A	2	**-13.15 ± 0.32**	**-13.25 ± 0.46**	**-13.14 ± 0.27**
	4	**-26.09 ± 0.39**	**-27.07 ± 2.35**	**-26.49 ± 0.51**
	6	**-39.29 ± 0.36**	-40.33 ± 1.44	-40.45 ± 1.40
	8	**-52.81 ± 1.30**	**-53.96 ± 1.67**	**-56.17 ± 4.35**
Network B	2	**-15.96 ± 0.65**	**-16.27 ± 0.67**	-17.41 ± 2.06
	4	**-33.60 ± 2.02**	**-33.58 ± 2.87**	**-33.18 ± 1.56**
	6	**-52.24 ± 2.26**	**-53.85 ± 5.13**	**-52.86 ± 4.00**
	8	**-68.62 ± 3.62**	**-69.36 ± 6.28**	-75.20 ± 3.56
Network C	2	**-17.33 ± 0.25**	**-17.23 ± 0.33**	**-17.83 ± 1.37**
	4	**-34.67 ± 0.62**	**-36.07 ± 3.76**	**-34.91 ± 0.85**
	6	**-52.71 ± 0.78**	**-53.62 ± 2.64**	-53.90 ± 1.37
	8	**69.88 ± 1.20**	**-71.20 ± 2.67**	-74.25 ± 2.53
Win95pts	2	**-65.64 ± 13.68**	**-211.43 ± 469.26**	-207.70 ± 471.30
	4	**-125.68 ± 13.49**	**-134.46 ± 17.50**	-4067.57 ± 614.37
	6	**-564.42 ± 384.45**	**-1027.76 ± 1440.54**	-6289.36 ± 1197.70
	8	**-1258.92 ± 367.16**	**-1395.12 ± 1016.51**	-7915.30 ± 1678.76
Insurance	2	**-36.20 ± 3.49**	**-37.13 ± 4.56**	-39.07 ± 3.75
	4	**-70.11 ± 5.83**	-82.21 ± 9.90	-91.77 ± 10.10
	6	**-105.18 ± 14.81**	-117.19 ± 14.62	-427.69 ± 371.47
	8	**-142.77 ± 15.25**	-163.98 ± 11.54	-722.26 ± 773.90
Hailfinder	2	**-83.50 ± 5.13**	-87.98 ± 6.71	**-85.27 ± 4.78**
	4	**-159.91 ± 8.63**	-242.77 ± 106.83	**-157.78 ± 9.44**
	6	**-274.13 ± 15.38**	**-277.02 ± 23.71**	-762.74 ± 662.30
	8	**-350.56 ± 28.82**	-379.61 ± 29.37	-1111.45 ± 1054.07
Hepar2	2	**-78.70 ± 0.32**	-79.19 ± 0.48	-80.94 ± 1.65
	4	**-157.60 ± 0.65**	-158.91 ± 1.64	-161.76 ± 1.94
	6	**-233.77 ± 1.67**	-236.04 ± 2.36	-252.35 ± 5.98
	8	**-232.18 ± 2.48**	**-234.58 ± 4.12**	-239.96 ± 5.49

Table 4.3 indicates that, for Network A, OSI and DOSI tie with exact inference when k is set to 2, 4, and 6, and OSI ties with exact inference when k is set to 8. For Network B, while the results of our algorithms are close to those of exact inference, both OSI and DOSI are outperformed by exact inference. For Network C, OSI and DOSI tie with exact inference when k is set to 2, 4, and 6. For all other networks, both OSI and DOSI outperform MBE and SLS.

4.5 Comparison Against Modifications of DOSI

The results of the comparison between DOSI-Comm and DOSI-Comp are shown in Table 4.5. For all networks, DOSI performs either equivalently or better than DOSI-Comp and DOSI-Comm. For Network A, all algorithms tie statistically when k is set to 2, 4, and 8. For Network B, all algorithms tie statistically when k is set to 4 and 6 while DOSI and DOSI-Comm tie statistically when k is set to 2 and 8. For Network C, all algorithms tie statistically for best when k is set equal to 2 and 4 while DOSI and DOSI-Comm tie statistically when k is set to 6 and 8. OSI outperforms all other approximate algorithms when k is set to 8. For Win95pts, DOSI and DOSI-Comm tie statistically for best for all values of k. For Insurance, DOSI and DOSI-Comm tie statistically for best when k is set to 2. For Hailfinder, DOSI and DOSI-Comp tie statistically for best when k is set to 2 and 4 while DOSI and DOSI-Comm tie statistically for best when k is set to 6. For Hepar2, DOSI and DOSI-Comm tie statistically for best when k is set equal to 8.

4.6 Comparison of Overlap Structures

The results for our comparison of the various overlap structures is presented in Table 4.6. For all networks, the Markov blanket architecture performs either equiva-

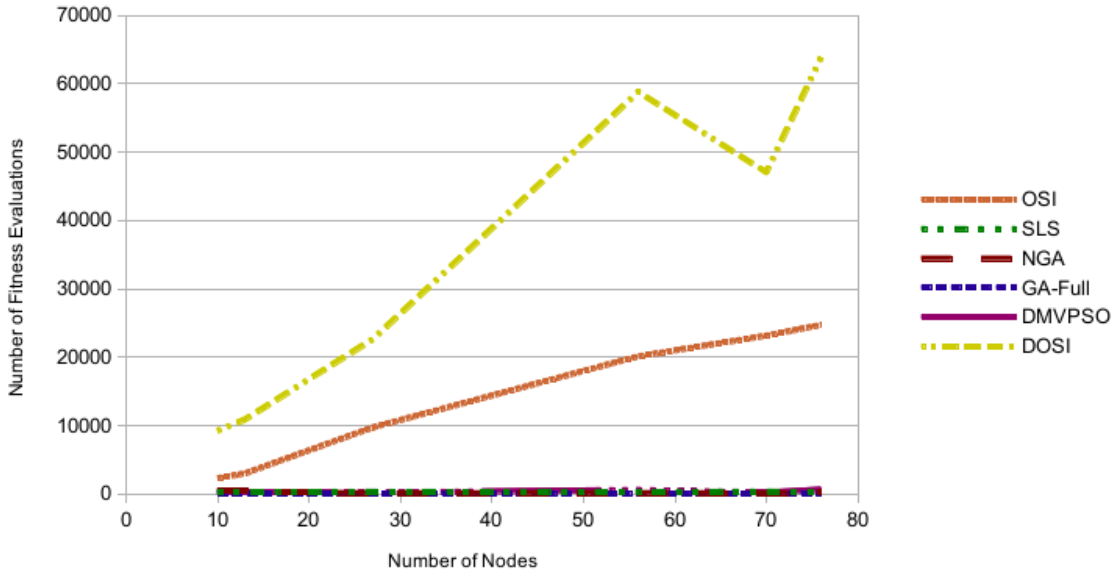


Figure 4.4: Number of Fitness Evaluations

Table 4.6: OSI results for different sub-swarm architectures.

	Markov	Clique	Parents	Random
Insurance	**-11.79 ± 1.37**	**-12.44 ± 1.63**	**-12.44 ± 1.53**	-13.27 ± 1.77
Win95	**-7.54 ± 2.08**	-8.64 ± 2.23	**-7.51 ± 1.79**	-8.96 ± 4.67
Hailfinder	**-35.64 ± 1.92**	-334.49 ± 368.11	**-66.64 ± 147.1**	**-37.35 ± 1.93**
Hepar2	**-16.37 ± 0.03**	-17.77 ± 1.81	-17.49 ± 2	-25.31 ± 3.29

lently or better than the other architectures. For the Insurance network, the Markov blanket architecture ties statistically with both the Clique and Parents architecture. For the Win95 network, the Markov blanket architecture ties with the Parents architectures. For the Hailfinder network, the Markov blanket architecture ties with the Parents and Random architectures. For the Hepar2 network, the Markov blanket architecture outperforms all other architectures.

4.7 Discussion

The paired t-tests on the sum of the log likelihoods indicate that both OSI and DOSI performed better than the other approximate methods for nearly all values of k. These results show that our algorithms outperform the other methods for all networks containing more than 15 nodes. This indicates that both OSI and DOSI have an advantage when used to perform inference on more complex networks. These results support our hypothesis that OSI and DOSI would outperform competing approaches for most experiments. Since multiple swarms learn the state assignments for a single variable, our approach ensures greater exploration of the search space. Through competition, we ensure that the best variable state assignments found by the swarms are used in the final k explanations.

The results shown in Table 4.5 indicate that, for all networks, DOSI performs either equivalently or better than the modified versions of DOSI and for all networks containing more than 15 nodes DOSI outperformed either one or both of these alternative implementations, again reinforcing the advantage of our approach in more complex networks. These results support our the hypothesis, that removing the competition or communication mechanism from DOSI would cause a drop in performance for some of the experiments.

The results in Table 4.6 indicate that, for all networks the Markov blanket architecture performs either equivalently or better than the other architectures. These results support the hypothesis that the performance of OSI is tied to the structure of the sub-swarms, and that the conditional dependencies between variables play an important role when selecting an optimal overlap structure. Specifically, these results indicate that the increased performance obtained by OSI and DOSI is due to the representation of each swarm being based on the Markov blankets and the corre-

sponding communication and competition that occurs between overlapping swarms. Recall that each variable X_i is conditionally independent of all other variables in the network given its Markov blanket. By assigning each node's swarm to a Markov blanket we ensure that the swarm learns the state assignments for all variables upon which that node may depend.

The results in Table 4.2 and 4.4 also indicate that the OSI algorithm outperforms DOSI for many experiments. We believe that this is due to the communication delay that results from DOSI being truly distributed. Because the DOSI algorithm requires several iterations of the communication procedure for the swarms to reach consensus with respect to the k most probable state assignments for each variable, the fitness evaluations of a particle may be inaccurate prior to reaching consensus. If consensus has not been reached, then some swarms will not have received updated state assignments from swarms outside of its Markov blanket, so the fitness evaluations performed in this swarm will rely on an outdated set of state assignments.

While OSI and DOSI appear to outperform the other methods in terms of the log likelihoods of solutions found, these algorithms require many more fitness evaluations than the other approaches. Figure 4.4 indicates that, while the number of nodes in the network has little effect on the number of fitness evaluations required by the competing algorithms, the number of fitness evaluations required by the OSI and DOSI algorithms is higher for networks with a large number of nodes. This is because our algorithms create a separate swarm for each of the nodes in the network, causing the number of swarms and the number of fitness evaluations to increase with the number of nodes.

CHAPTER 5

PARTIAL ABDUCTIVE INFERENCE IN BAYESIAN NETWORKS

While the goal of full abductive inference is to find the most probable states for all variables in the network, the goal of partial abductive inference is to find the most probable states for a subset of the variables in the network. This kind of inference is useful when only a small number of variables are of interest. Unfortunately, partial abductive inference is an even more difficult problem than full abductive inference, since inference must be performed to even evaluate a candidate solution to the problem.

To demonstrate the validity of applying OSI to problem of partial abductive inference we developed both distributed and centralized OSI-based algorithms for partial abductive inference and we compared our algorithms to existing approaches.

5.1 Partial Abductive Inference via OSI

Like the full abductive inference algorithm, our approach to partial abductive inference uses the DMVPSO algorithm [44] and is based on the OSI methodologies described in [15] and [53]. With this method, we assign a swarm to each node in the explanation set and a swarm to each node in the explanation node's Markov blanket.

Similar to full abductive inference, a global set of state assignments A is maintained across all swarms and is used for inter-swarm communication. Each particle's object parameters contain only state assignments for variables in the explanation set. Thus each particle represents a partial state assignment for the explanation set. The quality of each state assignment x_E for the explanation set is determined by the log

likelihood of that assignment given the evidence x_O as shown below:

$$L(x_E) \quad = \quad \log(P(x_E|x_O)) = \log\left(\sum_{x_R} P(x_E, x_R|x_O)\right)$$

This calculation can be computationally expensive, since it requires inference to be performed over the network. Because the problem of performing inference is NP-Hard, we use the likelihood weighting algorithm to evaluate the quality of a state assignment [84]. We chose likelihood weighting because its parameters can be set such that it provides a bound on error with some probability, but other inference methods could be used here, including exact methods such as Variable Elimination. In likelihood weighting, each sample s generated is associated with a weight value w_s, which is set to the product of the probability of the evidence variables given the states assigned to their parents in the sample:

$$w_s = \prod_{x \in x_O} P(x|\mathrm{Pa}(x))$$

Our algorithm begins by generating m samples using likelihood weighting given x_O as evidence. These samples are used to compute the approximate log likelihood (denoted L_a):

$$L_a(x_E) = \log\left(\sum_{s \in M} \frac{W(x_E, s)}{w_s}\right)$$

where M is the set of samples generated by likelihood weighting, w_s is the weight of sample s, and $W(B_p, s)$ is defined as,

$$W(x_E, s) = \begin{cases} w_s & x_E \in s \\ 0 & \text{otherwise} \end{cases}$$

As with OSI for full abductive inference, fitness evaluation requires that a set of state assignments $B_p = \{\beta_1, \ldots, \beta_k\}$ be constructed using the state of the particle p and the set of global state assignments A. The process of constructing B_p is identical to the process used in OSI for full abductive inference. We use B_p to calculate the fitness of each particle,

$$f(p) = \sum_{\beta_i \in B_p} L_a(\beta_i)$$

Thus the fitness of particle p is the sum of the approximate log likelihoods of the assignments in A when the value assignments encoded in B_p are substituted into A.

After each iteration of the algorithm, swarms that share a node in the network compete to determine which state is assigned to the node in each assignment $\alpha_m \in A$. This process is the same as for abductive inference and is shown in Algorithm 4.2.

5.1.1 Computational Complexity

We first describe 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.

- **Fitness Evaluation**– First, we determine the complexity of evaluating the fitness of an individual particle. Let F be the computational complexity specific to the fitness function used to evaluate each particle. During fitness evaluation we compute the log likelihood for k candidate explanations, where giving a total fitness function complexity of $\mathcal{O}(kF)$.

- **Optimization Algorithm**– Next, we determine the complexity of the underlying PSO algorithm. We assume that the algorithm has $q = |P|$ individuals, where P is the set of all the individuals in the population. During each iteration,

an individual, learning the state assignment for n variables in the worst case, 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 $\mathcal{O}(kF)$. This is done q times, once for for each individual; therefore, assuming the algorithm performs e iterations, the total complexity is $\mathcal{O}(kFeq)$.

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 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 $\mathcal{O}(kFeq)$. Since this is done $s = |\boldsymbol{S}|$ times, the total complexity of this step is $\mathcal{O}(skFeq)$.

- **Competition**– In OSI, the competition step 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 competing individuals. Note that there are n variables. In the worst case, all swarms will overlap over every variable, resulting in n competing subpopulations, while each fitness evaluation has a complexity of $\mathcal{O}(kF)$. Therefore, the total complexity of the competition step in OSI is $\mathcal{O}(n^2kF)$.

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

$$\mathcal{O}(\text{OSI}) = \mathcal{O}(m(skFeq + n^2kF))$$

$$= \mathcal{O}(mskFeq + n^2kF)$$

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

$$\mathcal{O}(\text{OSI}) = \mathcal{O}(mskFeq + n^2kF)$$

Based on these results, we can see that OSI will almost always be more computationally complex due to the n^2 factor caused by the competition phase. However, in practice, the worst case $\mathcal{O}(n^2kF)$ complexity of the competition phase will be rare, and will only occur when all variables in the network share a Markov blanket.

5.2 Partial Abductive Inference via DOSI

In this section we present our modification of the OSI algorithm for partial abductive inference that eliminates the need for a set of global state assignments to be maintained across all swarms. As with distributed full abductive inference, each swarm s maintains a set of the most probable personal state assignments found by that swarm, denoted as A_s. The most probable assignments learned by each swarm are communicated to the other swarms and inserted into the swarm's personal state assignment through a periodic communication mechanism. At each iteration of the algorithm, the swarm associated with a given node n will hold a competition between all swarms that share n to determine the most probable state assignment for n.

The fitness calculation and the calculation of $B_{s,p}$ for a given swarm s and particle p is identical to the calculation of $B_{s,p}$ used in DOSI for full abductive inference. We

use $B_{s,p}$ to calculate the fitness of each particle as follows

$$f(p) = \sum_{\beta_i \in B_{s,p}} L_a(\beta_i)$$

Thus the fitness of particle p in a swarm s is the sum of the approximate log likelihoods of the assignments in α_s when the state assignments encoded in p are substituted into α_s.

The *ShareStates* method in this approach is identical to the *ShareStates* method described for full abductive inference via DOSI (Algorithm 4.6) except that only state assignments for nodes within the explanation set are shared. Similarly the competition between nodes remains the same as the competition used for full abductive inference (Algorithm 4.2) except that swarms only compete over nodes within the explanation set. These modifications, combined with the use of likelihood weighting for fitness evaluation, are the primary differences between this approach and full abductive inference via DOSI.

5.3 Experimental Design

We compare partial abductive inference via OSI and DOSI against four other partial abductive inference algorithms:

- MBE: Mini bucket elimination algorithm proposed in [59].
- DMVPSO: Discrete multi-valued PSO algorithm proposed in [69].
- GA-Part: Genetic algorithm for partial abductive inference proposed in [67].
- SA: Simulated annealing algorithm proposed in [62].

For these experiments, we modified DMVPSO proposed in [69] so that it can be used to perform partial abductive inference. As with OSI, our modified DMVPSO uses likelihood weighting to approximate the log likelihood for fitness evaluation. To compare these algorithms, we used the same networks as described in Figure 4.3 and Table 4.1. Nodes were randomly selected for inclusion in the explanation set with a 25% probability.

For our experiments we ran each algorithm on each of the networks. We evaluated all algorithms with k set to 2, 4, 6, and 8 respectively. For all of the algorithms, initial populations were generated using a forward sampling process. In every experiment, the number of particles in each swarm was set to 20. For the genetic algorithms, the population size was set to 20. We ran all algorithms until convergence. We averaged the log likelihoods of the solutions found by the various algorithms over 10 runs and compared the solutions using a paired t-test to evaluate significance. For the t-test the confidence interval was 95%. We also measured the number of fitness evaluations performed by each of the algorithms to compare the computational complexity of each approach. We hypothesize that OSI and DOSI will outperform competing approaches in terms of log-likelihood for the more complex networks. Additionally, we hypothesize that the performance of DOSI will not significantly differ from that of OSI for most networks.

For networks A, B, and C, the MBE algorithm was used to compute the exact solution for the partial abductive inference problem. For all other networks MBE was used to find an approximate solution with parameters m and i set to 2 and 3 respectively.

5.4 Results

In Tables 5.1 and 5.2 we show the average sum of the log likelihoods for each algorithm and each value of k. Bold values indicate that the corresponding algorithm's performance is statistically significantly better than the other algorithms for the network given the corresponding value for k. Values for algorithms that tie statistically for best are also bolded.

For all networks other than Network A, OSI and DOSI tie statistically with or outperform MBE. For Networks A, B, and C, both OSI and DOSI find the optimal solution for all values of k with one exception. In Network A, when k is set to 2 neither OSI or DOSI find the exact solution.

For Network A, OSI, DOSI, and DMVPSO tie statistically when k is set to 2. Otherwise, OSI and DOSI have the best performance. For Network B, OSI, DMVPSO, and DOSI tie statistically for best for all values of k. For Network C, OSI, DMVPSO, and DOSI tie statistically when k is set to 4 while OSI and DOSI have the best performance otherwise. For Win95pts, and Insurance both DOSI and OSI have the best performance for all values of k. For Hailfinder, OSI, DMVPSO, and DOSI tie statistically for best for all values of k. For Hepar2, OSI and DOSI tie statistically when k is set to 2, 6, and 8. Otherwise, OSI has the best performance.

In Figure 5.1 we compare the number of fitness evaluations required by each of the algorithms with respect to the number of nodes in the network. We find a very similar performance trend to Figure 4.4 for similar reasons.

Table 5.1: Comparison with other approaches

Network	k	OSI	GA-Part	DMVPSO	SA	MBE
Network A	2	**-3.20 ± 0.00**	-7.70 ± 1.92	**-3.20 ± 0.00**	-7.70 ± 2.58	**-2.91**
	4	**-8.39 ± 0.00**	-14.59 ± 1.82	-8.92 ± 0.72	-11.86 ± 3.59	**-8.39**
	6	**-13.95 ± 0.00**	-21.89 ± 2.17	-14.62 ± 0.84	-21.24 ± 2.59	**-13.95**
	8	**-23.57 ± 0.00**	-24.95 ± 1.49	-23.57 ± 0.00	-25.01 ± 1.44	**-23.57**
Network B	2	**-2.87 ± 0.00**	-4.37 ± 0.65	**-2.87 ± 0.00**	-4.84 ± 0.68	**-2.87**
	4	**-7.55 ± 0.00**	-9.03 ± 0.94	**-7.55 ± 0.00**	-9.10 ± 1.30	**-7.55**
	6	**-12.40 ± 0.00**	-14.33± 0.54	**-12.40 ± 0.00**	-14.38 ± 0.76	**-12.40**
	8	**-18.22 ± 0.00**	-18.83 ± 0.60	**-18.22 ± 0.00**	-18.85 ± 0.43	**-18.22**
Network C	2	**-5.17 ± 0.00**	-7.26 ± 1.79	-5.17 ± 6.81	-8.01 ± 2.15	**-5.17**
	4	**-11.47 ± 0.00**	-15.77 ± 4.33	**-11.47 ± 0.00**	-18.51 ± 4.54	**-11.47**
	6	**-18.20 ± 0.00**	-24.40 ± 2.83	-18.41 ± 0.26	-37.80 ± 11.32	**-18.20**
	8	**-25.15 ± 0.00**	-33.29 ± 3.74	-26.32 ± 1.36	-58.56 ± 14.54	**-25.15**
Win95pts	2	**-5.29 ± 0.00**	-9.96 ± 2.98	-5.62 ± 0.45	-601.36 ± 310.82	-753.65
	4	**-12.45 ± 0.00**	-20.23 ± 2.47	-13.26 ± 0.97	-1943.27 ± 381.27	-2952.21
	6	**-20.48 ± 0.00**	-32.14 ± 3.82	-21.89 ± 1.58	-3135.72 ± 468.31	-2961.54
	8	**-28.69 ± 0.00**	-43.14 ± 5.16	-32.22 ± 3.32	-4403.11 ± 886.28	-5904.43
Insurance	2	**-6.47 ± 1.24**	-1045.61 ± 380.91	-9.86 ± 2.19	-1341.29 ± 310.79	-22.52
	4	**-13.05 ± 0.52**	-2313.46 ± 811.42	-28.85 ± 17.83	-2535.85 ± 380.35	-42.75
	6	**-20.29 ± 0.35**	-3285.61 ± 864.35	-574.56 ± 760.88	-4318.95 ± 311.18	-62.97
	8	**-27.88 ± 0.49**	-3963.88 ± 855.26	-298.07 ± 346.66	-5659.35 ± 516.60	-87.79
Hailfinder	2	**-14.49 ± 0.52**	-607.45 ± 574.35	**-14.65 ± 0.76**	-972.83 ± 353.91	-1484.99
	4	**-31.32 ± 2.39**	-929.10 ± 821.42	**-31.97 ± 4.24**	-2464.16 ± 352.25	-2969.98
	6	**-49.79 ± 6.23**	-1456.29 ± 417.35	**-47.66 ± 4.22**	-4101.11 ± 381.97	-4454.96
	8	**-72.93 ± 4.33**	-1694.31 ± 968.11	**-71.91 ± 3.14**	-5441.21 ± 354.90	-5939.95
Hepar2	2	**-8.12 ± 0.42**	-20.68 ± 1.57	-9.47 ± 0.93	-609.28 ± 308.44	-25.03
	4	**-17.44 ± 0.76**	-39.46 ± 4.09	-19.60 ± 2.01	-157.19 ± 11.44	-49.81
	6	**-27.49 ± 1.03**	-57.81 ± 5.53	-32.77 ± 2.07	-2112.15 ± 1336.86	-73.46
	8	**-36.87 ± 1.34**	-80.85 ± 7.33	-42.97 ± 2.18	-3754.73 ± 1094.14	-100.14

5.5 Discussion

The paired t-tests on the sum of the probabilities indicate that OSI and DOSI performed either equal to or better than the other methods for most values of k on nearly all networks. While the OSI approach does not appear to have an advantage over DMVPSO when used on Hailfinder and Network B, we can see that it outperformed the other methods on all other networks for most values of k. We believe that our algorithms were not able to outperform DMVPSO approaches when used on Hailfinder because this network has many nodes with conditional probability distributions that are essentially uniform. This may have allowed DMVPSO to perform

Table 5.2: Comparison with DOSI

Network	k	OSI	DOSI
Network A	2	**-3.20 ± 0.00**	**-3.20 ± 0.00**
	4	**-8.39 ± 0.00**	**-8.39 ± 0.00**
	6	**-13.95 ± 0.00**	**-13.95 ± 0.00**
	8	**-23.57 ± 0.00**	**-23.57 ± 0.00**
Network B	2	**-2.87 ± 0.00**	**-2.87 ± 0.00**
	4	**-7.55 ± 0.00**	**-7.55 ± 0.00**
	6	**-12.40 ± 0.00**	**-12.40 ± 0.00**
	8	**-18.22 ± 0.00**	**-18.22 ± 0.00**
Network C	2	**-5.17 ± 0.00**	**-5.17 ± 0.00**
	4	**-11.47 ± 0.00**	**-11.47 ± 0.00**
	6	**-18.20 ± 0.00**	**-18.20 ± 0.00**
	8	**-25.15 ± 0.00**	**-25.15 ± 0.00**
Win95pts	2	**-5.29 ± 0.00**	**-5.29 ± 0.00**
	4	**-12.45 ± 0.00**	**-12.45 ± 0.00**
	6	**-20.48 ± 0.00**	**-20.48 ± 0.00**
	8	**-28.69 ± 0.00**	**-28.69 ± 0.00**
Insurance	2	**-6.47 ± 1.24**	**-6.68 ± 1.22**
	4	**-13.05 ± 0.52**	**-13.60 ± 1.16**
	6	**-20.29 ± 0.35**	**-20.59 ± 0.92**
	8	**-27.88 ± 0.49**	**-27.84 ± 0.53**
Hailfinder	2	**-14.49 ± 0.52**	**-14.95 ± 1.03**
	4	**-31.32 ± 2.39**	**-31.31 ± 1.70**
	6	**-49.79 ± 6.23**	**-49.80 ± 3.30**
	8	**-72.93 ± 4.33**	**-72.03 ± 4.24**
Hepar2	2	**-8.12 ± 0.42**	**-8.48 ± 0.58**
	4	**-17.44 ± 0.76**	-19.80 ± 0.85
	6	**-27.49 ± 1.03**	**-28.15 ± 1.41**
	8	**-36.87 ± 1.34**	**-38.42 ± 1.49**

well given that, for many of the nodes, the states chosen did not affect the solution quality. For future experiments, it may be worthwhile to control for this by ensuring that such nodes are not included in the explanation set.

Consistent with our lesion studies on full abductive inference, we believe that the increased performance of OSI and DOSI is due to each swarm being associated with the Markov blanket of a single node and the resulting communication and competition between overlapping swarms, as in the full abductive inference case. This is in contrast to DMVPSO, where a single swarm learns the state assignment for all variables. This can lead to a more local exploration around the solution found by the swarm's global

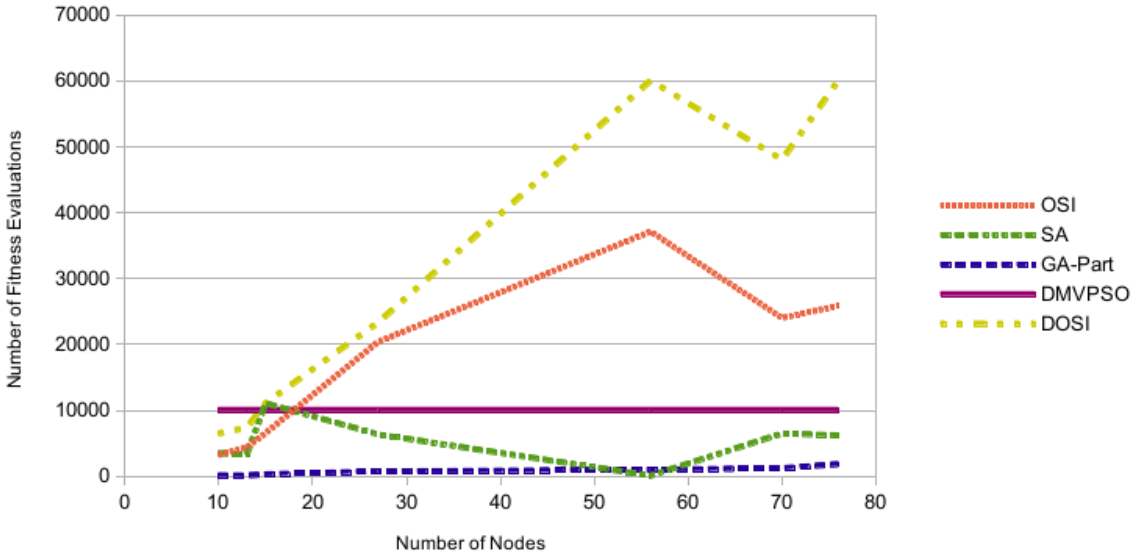


Figure 5.1: Number of Fitness Evaluations

best. Also, because each swarm optimizes over the all variables, there is an increased possibility of neglecting a potentially good solution for a specific component of the solution vector for the global best particle.

We also note that, MBE performed poorly on many of the larger networks when compared to OSI, DOSI, and DMVPSO. This is likely due to the fact that MBE becomes a greedy approximation for abductive inference when the factor maximization is performed over the smaller-sized mini-buckets rather than the complete buckets used for exact inference. Such a greedy approach does not provide the necessary exploration needed to find good state assignments for these larger networks, when compared to the swarm based algorithms.

DOSI was found to outperform all approaches not based on OSI, and DOSI tied statistically with OSI for most networks and values for k. This validates our hypothesis that the performance of DOSI would not significantly differ from that of OSI for most networks. This result indicates that DOSI provides an effective distributed al-

ternative to traditional OSI for the problem of partial abductive inference. While OSI and DOSI appear to outperform the other methods in terms of the log likelihood of solutions found, these methods require many more fitness evaluations than the other approaches. However, the correlation between the number of nodes and number of fitness evaluations is not as strong in the case of partial abductive inference since the number of required swarms is tied to the size of the explanation set rather than the number of nodes in the network.

CHAPTER 6

PARAMETER ESTIMATION

In this chapter, we explore the problem of learning the parameters for Bayesian networks with latent variables. We describe an OSI algorithm for latent variable parameter estimation in Bayesian networks. To evaluate the improvement provided by OSI, we present a traditional single-swarm approach to parameter estimation and then compare our multi-swarm approach to single-swarm PSO and several other existing approaches to parameter estimation.

In Bayesian networks, parameter estimation is the problem of discovering conditional probabilities of random variables when data for these random variables is missing or incomplete. Traditionally, this problem has been approached using Markov chain Monte-Carlo techniques or expectation maximization. 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.

6.1 Single-Swarm Particle Swarm Optimization

To our knowledge, PSO has not been applied to parameter estimation in Bayesian networks; therefore we have developed both a single swarm and an OSI-based approach to the problem. This 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 $\mathbf{x}$ is a d-dimensional vector of real numbers where $\mathbf{x} \in [0, 1]^d$, 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 $\mathbf{D}$ we can estimate any joint probability over the observed variables as

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

where

$$z = \prod_{i=1}^{n} |Val(X_i)|$$

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:

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

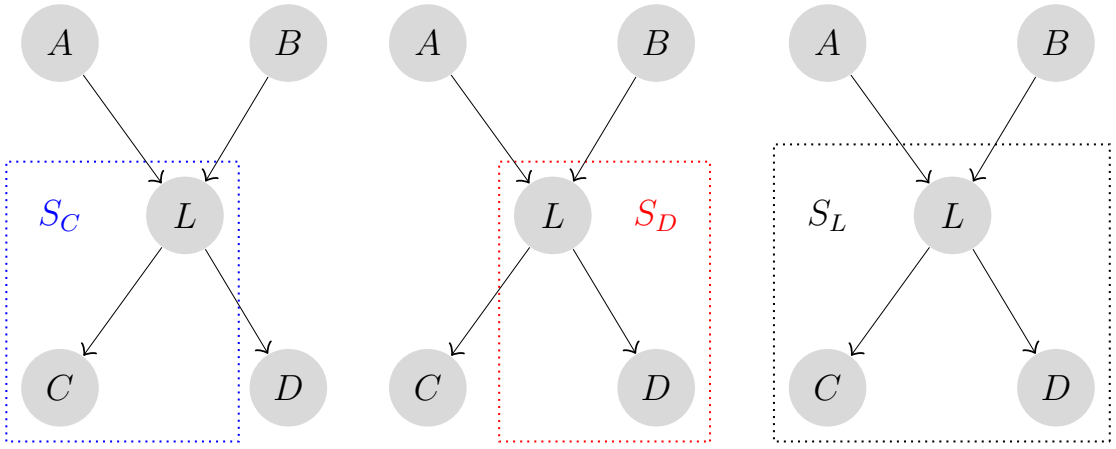
Because these probabilities can be computed directly, PSO will not optimize the parameters for observed variables with no latent parents.

Once the parameters have been computed, each particle's fitness is evaluated. Since we are learning parameters for networks containing latent variables, the states of these variables will be missing from the data, and the latent variables must be marginalized out in order compute the log likelihood of the data given the network. To compute the fitness of a set of parameters, we use variable elimination to calculate the log likelihood of the data $\mathbf{D}$ given the parameters Θ.

$$L(\mathbf{D}|\Theta) = \log P(\mathbf{D}|\Theta). \tag{6.1}$$

We chose variable elimination instead of an approximate sampling algorithm, because we found that the probability of the data given the model was always very close to zero and an extremely large number of samples would be required to obtain non-zero probabilities, thereby offsetting any advantage that would be gained from using an approximate sampling algorithm.

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 in traditional PSO.



(a) Nodes learned by S_C (b) Nodes learned by S_D (c) Nodes learned by S_L

Figure 6.1: Swarm assignment example

6.2 Overlapping Swarm Intelligence

We have 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 6.1 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 6.1(a) shows that parameters will be learned by swarm S_C, which covers nodes L and C; 6.1(b) shows that parameters will be learned by swarm S_D, which covers nodes L and D; and Figure 6.1(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.

The pseudocode for our approach is shown in Algorithm 6.1. The algorithm is split into two main loops that are executed repeatedly until some termination criterion is met. 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 $\mathbf{x}$ is defined by a d-dimensional vector of continuous values where $\mathbf{x} \in [0, 1]^d$, 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 single-swarm 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 $\text{MB}(s)$ 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 $\Theta_p = \{\theta_{p,v} | v \in \text{MB}(s)\}$ 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:

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

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

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

Algorithm 6.2 Generate Parameters

Procedure Get-Params(
G // A Bayesian network structure
$\mathbf{x}$ // particle position vector
Θ_g // set of global parameter estimates
)

1: Let $\Theta_{g,p}$ be the set of personal parameters
2: Estimate parameters for observed variables and add them to $\Theta_{g,p}$
3: **for** position value $x_{a|\mathrm{Pa}(A)} \in \mathbf{x}$ **do**
4: $P(a|\mathrm{Pa}(A)) = \frac{x_{a|\mathrm{Pa}(A)}}{\sum_{a' \in A} x_{a'|\mathrm{Pa}(A)}}$
5: Add $P(a|\mathrm{Pa}(A))$ to $\Theta_{g,p}$
6: **end for**
7: **return** $\Theta_{g,p}$

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 $\mathbf{D}$ in Figure 6.1(a)) 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 6.1(c), the swarms associated with C and $\mathbf{D}$ would compete to determine which parameter values are assigned to the node L.

Algorithm 6.1 Overlapping Swarm Intelligence for Parameter Estimation

Procedure OSI-PE(
  G // A Bayesian network structure
  D // training data
  V // set of variables with missing parameters
)

Randomly initialize set of global parameters Θ_g
Initialize particles in each swarm
repeat
  for each swarm s **do**
    for each particle, $p \in s$ **do**
      $\Theta_{g,p} \leftarrow$ Get-Params$(G, \mathbf{x}_p, \Theta_g)$
      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 $V \in \mathbf{V}$ **do**
    Let the set of swarms $S = \{s | V \in \text{MB}(s)\}$
    for each parameter $z \in \theta_{g,x}$ **do**
      $z \leftarrow Compete(S, z)$
    end for
  end for
until termination criterion is met
return Θ_g

6.2.1 Computational Complexity

Here we describe the computational complexity of OSI for parameter estimation 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 pseudo code shown in Algorithm 6.1.

- **Fitness Evaluation**– First, we determine the complexity of evaluating the fitness of an individual particle. Let $p = h \times |\theta_{max}|$ be the total number of parameters to be learned, in the worst case, where h is the number of factors in the network, and θ_{max} factor with the largest number of parameters. During fitness evaluation we compute the log likelihood of the data given the model, which requires performing variable elimination d time, where $d = |\mathbf{D}|$. Since, variable elimination has a time complexity of $\mathcal{O}(p)$ [12], the fitness function complexity is $\mathcal{O}(dp)$.

- **Optimization Algorithm**– Next, we determine the complexity of the underlying PSO algorithm. We assume that the algorithm has $q = |P|$ individuals, where P is the set of all the individuals in the population. During each iteration, an individual, optimizing p parameters, in the worst case, has its position and fitness updated. These two steps are done sequentially, with updating the position having a complexity of $\mathcal{O}(p)$ and while evaluating the fitness is $\mathcal{O}(dp)$. This is done q times, once for for each individual; therefore, assuming the algorithm performs e iterations, the total complexity is $\mathcal{O}(dpeq)$.

  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 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 $\mathcal{O}(dpeq)$. Since this is done $s = |\boldsymbol{S}|$ times, the total complexity of this step is $\mathcal{O}(sdpeq)$.

- **Competition**– In OSI, the competition step is used to find the optimal set of values for the global parameters. This is done by iterating over all the parameters and then comparing the fitness of the competing individuals. Note that there are p parameters. In the worst case, all swarms will overlap over every variable, resulting in s competing subpopulations, while each fitness evaluation has a complexity of $\mathcal{O}(dp)$. Therefore, the total complexity of the competition step in OSI is $\mathcal{O}(sdp^2)$.

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

$$\begin{aligned}\mathcal{O}(\text{OSI}) &= \mathcal{O}(m(sdpeq + sdp^2)) \\ &= \mathcal{O}(msdpeq + msdp^2)\end{aligned}$$

Based on these results, we can see that OSI will almost always be more computationally complex due to the p^2 factor caused by the competition phase. However, the parameters m, e, and q in OSI can be set such that it becomes competitive with PSO.

6.3 Experimental Design

To test the performance of our algorithms, several experiments were performed using data generated from networks in the Bayesian Network Repository [79]. 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) [75, 12, 76, 70].

For these experiments, we chose networks with gradually increasing numbers of parameters to evaluate the effect of network complexity on performance. 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. We manually chose latent variables that had at least one parent and one child, and that met the overlap criteria described above. A cross-reference to the network information and latent variable configurations for the datasets is shown in Table 6.1. 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:

$$\text{Overlap} = \frac{1}{|S|} \sum_{s \in S} N(s)$$

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.

Table 6.1: 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, appbata, 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

For our algorithm, six particles were assigned to each sub-swarm. During preliminary testing, we found that assigning more than six particles to each swarm slowed the algorithm while providing little benefit to performance. 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 [82].

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 one third of the original data points that were sampled at random during each fitness calculation. The data was re-sampled during each fitness evaluation to ensure that all data-points in the training data would likely be used to evaluate particle fitness at some point in algorithm. 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. We chose 5×2 cross-validation instead of 10 fold cross-validation because of the faster execution of the former. 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. We hypothesize that, OSI will outperform single-swarm PSO for most of the networks, and PSO will never outperform OSI. We also hypothesize that, performing fitness evaluation relative to a sampled subset of the data will not have a significant impact on parameter quality for most networks.

6.4 Results

Table 6.2 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-O 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 other than OSI-S. However, while OSI and OSI-S were statistically significantly better

Table 6.2: Comparison against other approaches

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

than all the other algorithms on all networks, on the Alarm-I network, OSI was also statistically significantly than OSI-S. 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.

6.5 Discussion

The paired t-tests on the log likelihoods indicate that OSI and OSI-S perform 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. Rather than compute the expected sufficient statistics exactly, MCEM samples the incomplete data from the model using the current parameter estimates. This completed data set is then used to estimate the new parameters using the maximization step of EM. 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 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 state assignments 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.

While these results are encouraging, more work must be done to empirically verify the effect of conditional dependencies on the optimal overlap structure. 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.

CHAPTER 7

BAYESIAN NETWORK STRUCTURE LEARNING

While Bayesian networks can be developed by experts when the number of variables is small, the problem of defining a Bayesian network is often too complex for manual construction. In these cases, the Bayesian network structure must be learned from data. Chickering *et al.* have shown that the general problem of learning a Bayesian network model from data is NP-Hard [8]. Due to the complexity of learning a globally optimal network structure, several authors have developed approximate and heuristic algorithms for the problem. Many of these heuristics include machine learning techniques [9, 10, 11].

In this chapter, we explore the problem of structure learning in Bayesian networks. We describe an OSI algorithm for learning the structure of Bayesian network classifiers and we propose an OSI algorithm for more general structure learning, first in the presence of complete data and then in the context of latent variables.

7.1 Learning Bayesian Classifiers

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.

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 7.1.

$$m_{i,j} = \begin{cases} 1 & \text{if node } j \text{ is a parent of node } i \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

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. Since TAN learns the tree structure that maximizes the likelihood of the data given the model, this initial network will ensure that particles do not initially explore unproductive regions of the search space.

Each particle's position $\mathbf{x}$ is defined by a d-dimensional vector of binary values where $\mathbf{x} \in \{0, 1\}^d$, $d = 2n$, and n is the number of nodes in the network. The vector is of length $d = 2n$ because 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 7.1. In the first section of the figure, the representation 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

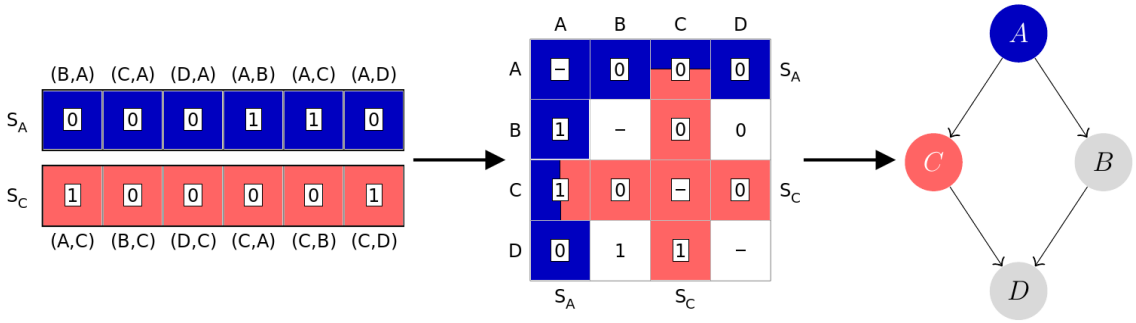


Figure 7.1: Representation example

learned by swarm S_C are shown in a different shade than the positions learned by swarm S_A. Entries in the matrix that are learned by both swarms are highlighted with both shades. 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 outgoing edges. This process is repeated until all cycles are eliminated.

For this problem, the overlap is defined by conflicts between the various swarms. In our approach, 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 lowest classification error 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 individual in the swarm with the value in the global solution.

Algorithm 7.1 Overlapping Swarm Intelligence for Classifier Learning

Procedure OSI-Learn-Classifier(
D // Training data
)

Initialize the global network G and adjacency matrix M
Initialize particles in each swarm
repeat
  for each swarm s **do**
    for each particle, $p \in s$ **do**
      Add edges of p to matrix M forming a new matrix M_p
      Remove cycles from M_p and construct Bayesian network 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 entry $m_{i,j} \in M$ **do**
    Let s_i be the swarm associated with node i
    Let s_j be the swarm associated with node j
    if s_i and s_j conflict over $M_{i,j}$ **then**
      $M_{i,j} \leftarrow Compete(s_i, s_j)$
      if s_i wins competition **:** seed s_j with $M_{i,j}$
      else seed s_i with $M_{i,j}$
    end if
  end for
until termination criterion is met
Construct Bayesian network G from the matrix M
return G

7.1.1 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 6.1. The second fitness function (OSI-BIC) is Bayesian information criterion:

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

The third fitness function (OSI-ACC) is the classification error of the network when used to classify the training data:

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

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:

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

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.

7.1.2 Computational Complexity

We first describe 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 pseudo code shown in Algorithm 7.1.

- **Cycle Removal**– Cycle removal can be performed in $\mathcal{O}(|V|^2)$ where V is the set of vertices. Let a be the number of entries in the adjacency matrix. Then, the $|V| = \sqrt{a}$ and the complexity of cycle removal is $\mathcal{O}(a)$.

- **Fitness Evaluation**– Next, we determine the complexity of evaluating the fitness of an individual particle. To do so, we approximate the fitness function complexity as $\mathcal{O}(aF)$, where a is number of entries in the adjacency matrix, and F the computational complexity specific to the fitness function used to evaluate each particle. When this is combined with cycle removal the time complexity becomes $\mathcal{O}(aF + a) = \mathcal{O}(aF)$

- **Optimization Algorithm**– Next, we determine the complexity of the underlying PSO algorithm. We assume that the algorithm has $q = |P|$ individuals, where P is the set of all the individuals in the population. During each iteration, an individual, with a variables in the worst case, has its position and fitness updated. These two steps are done sequentially, with updating the position having a complexity of $\mathcal{O}(a)$ and while evaluating the fitness is $\mathcal{O}(aF)$. This is done q times, once for for each individual; therefore, assuming the algorithm performs e iterations, the total complexity is $\mathcal{O}(aFeq)$.

  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 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 $\mathcal{O}(aeq)$. Since this is done $s = |\boldsymbol{S}|$ times, the total complexity of this step is $\mathcal{O}(saFeq)$.

- **Competition**– In OSI, the competition step 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 $\mathcal{O}(aF)$. Therefore, the total complexity of the competition step in OSI is $\mathcal{O}(2a^2F) = \mathcal{O}(a^2 \times F)$.

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

$$\begin{aligned}\mathcal{O}(\text{OSI}) &= \mathcal{O}(m(aFeqs + a^2F)) \\ &= \mathcal{O}(maFeqs + ma^2F)\end{aligned}$$

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

$$\begin{aligned}\mathcal{O}(\text{OSI}) &= \mathcal{O}(ma^2Feq + ka^2F) \\ &= \mathcal{O}(ma^2Feq)\end{aligned}$$

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

7.1.3 Experimental Design

To evaluate our algorithm, several experiments were performed using datasets from the UCI Machine Learning Repository [85]. These experiments focused on comparing our algorithm with the PSO algorithm discussed in [9], the Tree Augmented Naive Bayes algorithm [32], and the Greedy Thick Thinning algorithm [78]. 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 7.1.1.

For the OSI algorithms, five particles were assigned to each sub-swarm. This value was chose because, during preliminary testing, we found that a larger number of particles did not improve performance and slowed training time. 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 w was set to 0.7298. Eberhart and Shi empirically determined that these are good parameter choices for w, ϕ_1, and ϕ_2 [82].

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 chose 5×2 to ensure that both training and testing sets were large, and that each data point was used for both training and testing during each fold. 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 single-swarm PSO for most datasets and, when the proper fitness function is chosen, OSI will outperform both TAN and Greedy.

Table 7.1: 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

These experiments were performed using seven datasets downloaded from the UCI machine learning repository: 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 7.1.

7.1.4 Experimental Results

Tables 7.2 and 7.3 show the average classification accuracy for each algorithm and each dataset. The values presented in these tables do not reflect the fitness scores achieved by any of the population-based algorithms. 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 7.2 compares the classification accuracy of the swarm based approaches. For all datasets, either OSI-ACC 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 7.2: Comparison of classification accuracy for Swarm algorithms

Dataset	OSI-LL	PSO-LL	OSI-CMI	PSO-CMI	OSI-BIC	PSO-BIC	OSI-ACC	PSO-ACC
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 7.3: Comparison of classification accuracy against TAN and Greedy

Dataset	OSI-CMI	PSO-CMI	OSI-ACC	PSO-ACC	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

Table 7.3 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-ACC tie statistically for best. For the TicTacToe dataset OSI-ACC ties statistically with PSO-ACC for best. OSI-CMI outperforms all competing algorithm for every other dataset.

7.1.5 Discussion

The paired t-tests on the classification accuracy indicate that either OSI-ACC, 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-ACC 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-ACC 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 the 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 7.2 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-ACC 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.

7.2 Generative Structure Learning

The method proposed in the previous section can be modified to allow for generative Bayesian network structure learning. The pseudocode for this algorithm is shown in Algorithm 7.2. For this problem, we will use the Bayesian information criterion

(BIC) described in Equation 2.4 to evaluate network quality.

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

It can be shown that BIC decomposes into a sum of terms, one for each node X_i in the network [12]. This decomposability may reduce the need for overlap and competition in the OSI algorithm.

As with OSI for classifier learning, we associate a swarm with each node in the network but, for this problem, we will compare two different overlap architectures. In the first, each swarm will learn the incoming and outgoing edges associated with its corresponding node (similar to the classifier learning problem). In the second, each node's corresponding swarm will learn only the incoming edges associated with that node. We refer to the first architecture as OSI-F, while we refer to the second as OSI-N.

As in the previous algorithm, 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 7.1.

Each particle's position x_i is defined by a d-dimensional vector of binary values where $x_i \in \{0, 1\}^d$, $d = n$, and n is the number of nodes in the network. Each position value in the vector determines if a given variable in the network is a parent of the node. For a particle in the swarm associated with node i, the position vector encodes all elements in the 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.

Algorithm 7.2 OSI for Structure Learning with Complete Data

```
    Procedure OSI-Learn-Gen(
       D     // Training data
    )
 1: Randomly initialize the global network G and adjacency matrix M
 2: Initialize particles in each swarm
 3: repeat
 4:       for each swarm s do
 5:          for each particle, p ∈ s do
 6:             Add edges defined by 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} ∈ M do
19:          Let the s_i be the swarm associated with node i
20:          m_{i,j} ← value associated with best particle in s_i
21:       end for
22: until termination criterion is met
23: return M
```

To insert a substructure S_i for a node X_i into the network, all incoming edges for X_i are removed and the edges encoded in S_i are added to the network. This is done by inserting the row elements defined by a particle's position vector into the global adjacency matrix M. From this adjacency matrix, a Bayesian network is constructed.

When inserting the substructure of a particle into the network, cycles may be introduced. To prevent this, the cycle removal process described above is performed prior to fitness evaluation. Once cycles have been removed, we evaluate the fitness of the particle using BIC.

7.2.1 Experimental Design

For these experiments, we compared the above variations of OSI to a traditional single swarm PSO and the K2 structure learning algorithm. For these comparisons we used four datasets from the UCI Machine Learning Repository [85]: Vote, Car, Spect, and TicTacToe.

For the OSI algorithms, the number of particles in each swarm was set to three. This was done to ensure a manageable runtime for the population based approaches. The number of particles in the traditional PSO algorithm was set to three times the number of nodes, to ensure that all swarm algorithms had the same total number of particles. For both swarm-based algorithms ϕ_1 and ϕ_2 were set to 1.49618, while w was set to 0.7298.

To evaluate the performance of the algorithms, each data-set was divided into training and testing data sets using a 5×2 cross-validation procedure to ensure large training and test sets. The log-likelihoods of the best networks found by each algorithm were averaged over all runs. We performed pairwise comparisons of average log-likelihood using a paired student t-test for each pair of algorithms. For all t-tests we used a 95% confidence interval.

7.2.2 Experimental Results

The results for these experiments are shown in Table 7.4. Bold values indicate that the corresponding algorithms performance is statistically significantly better than the other algorithms on the data set.

For all data sets, we observe that, based on the paired t-tests on log-likelihood, OSI-F significantly outperforms both PSO and K2. Additionally, OSI-F outperforms OSI-N, for three of the four networks. For all networks, all particle based methods

Table 7.4: Average log-likelihoods for general structure learning

	OSI-N	OSI-F	PSO	K2
Vote	-2834.4	**-2638.7**	-2849.1	-4400.8
Car	-6870.2	**-6837.4**	-6949.6	-9940.1
Spect	-1758.3	**-1666.6**	-1767.9	-3345.7
TicTacToe	**-4880.5**	**-4884.3**	-4906.2	-7768.9

outperformed K2. For the Vote and Spect data sets, we observe that, based on the paired t-tests on log-likelihood, OSI-N and PSO tied statistically for best.

7.2.3 Discussion

The performance of OSI-F relative to OSI-N seems to indicate that the overlap and competition provide an advantage, even when there is little overlap between the sub-problems. Because several swarms learn the edges associated with each variable, OSI ensures greater exploration of the search space, and the inter-swarm competition ensures that the best sub-structure found by each swarm is included in the global network.

OSI-N outperformed single-swarm PSO on two of the four networks and never performed worse than single-swarm PSO. This indicates that OSI has an advantage over single swarm PSO, even when there is no overlap. By splitting the swarms over the nodes of the network, we reduce the possibility of neglecting a potentially good sub-structure for a specific component of the global networks complete structure. This advantage is present even when no overlap or competition is held between the swarms.

7.3 Structure Learning with Latent Variables

We have modified our approach to generative Bayesian network structure learning to allow for the learning of Bayesian networks with latent variables. This approach assumes that the number of latent variables is a user-specified parameter. As in Algorithm 7.2, we will use the Bayesian information criterion (BIC) to evaluate network quality, but since latent variables have been introduced into the problem, this scoring function will no longer be decomposable. As with the algorithm described in section 7.2, a swarm is associated with each node in the network and a global network structure is maintained for inter-swarm communication. The network structure is represented by an $n \times n$ adjacency matrix M and each element $m_{i,j}$ in the matrix is defined as shown in Equation 7.1.

To exploit the conditional dependencies introduced by the latent variables, each swarm in this algorithm learns the incoming edges for the Markov blanket of its associated node, as defined by the current global network structure. Because this overlap structure may change at each iteration, we refer to this algorithm as OSI with dynamic Markov blanket overlap (OSI-MB). For a swarm associated with some node i, each particle's position $\mathbf{x}_i$ is defined by a d-dimensional vector of binary values where $\mathbf{x}_i \in \{0,1\}^d$, $d = mn$, n is the number of nodes in the network, and $m = |\text{MB}(i)|$. For a particle in the swarm associated with node X_i, the position vector encodes all elements in the rows associated with Markov blanket of node X_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.

At the end of each iteration, the set of rows learned by a given swarm will be updated based on the Markov blanket of the swarm's corresponding node. During this process, position and velocity values that do not correspond to incoming edges

for the Markov blanket will be removed. If the corresponding Markov blanket has incoming edges that are not currently being optimized by the swarm, then random position and velocity values corresponding to these edges will be added to the particles of the swarm.

When evaluating the fitness of a given particle, the row elements defined by a particle's position vector are inserted into the global adjacency matrix M. From this adjacency matrix a Bayesian network is constructed and cycles are removed from the structure as described in section 7.1. Next, the parameters for each variable in the network are estimated using the OSI algorithm for parameter estimation described in Chapter 6. The fitness of the network is then evaluated by computing the BIC score of the network.

When multiple swarms learn the incoming edges for a given node, these swarms are said to overlap. After each iteration of the algorithm, overlapping swarms compete to determine which edges to include in the global network structure. This competition is held between the edges encoded in the personal best particles of each swarm. The edges resulting in the highest BIC score are selected for inclusion in the global adjacency matrix M.

7.3.1 Experimental Design

Several experiments have been performed to evaluate the performance of OSI for structure learning with latent variables. We compare OSI with dynamic Markov blanket overlap (OSI-MB) to both single-swarm PSO and OSI with a static overlap structure in which each node learns the associated incoming and outgoing edges (OSI-F). We also compare these algorithms to structural EM; a commonly used algorithm for structure learning in the presence of latent variables.

The settings for the various PSO-based algorithms are the same as described in section 7.2.1. For parameter estimation, the single-swarm PSO algorithm used structural EM, while the OSI algorithms used the parameter estimation algorithm described in Chapter 6. The OSI parameter estimation was run for six iterations during each fitness evaluation and competition was held every third iteration. Our goal was to run the parameter estimation algorithm for a small number of iterations to minimize the algorithm's runtime, and during preliminary testing we found that the above settings produced good results. When learning parameters for the final network, OSI was run until convergence.

As in section 7.2.1, the Vote, Car, Spect, and TicTacToe data-sets were used for evaluation. To evaluate the performance of the algorithms, each data-set was divided into training and testing data sets using a 5×2 cross-validation procedure to ensure large training and testing sets, and the log-likelihoods of the best networks were averaged. Pairwise comparisons of average log-likelihood were performed using a paired student t-test for each pair of algorithms. For all t-tests we used a 95% confidence interval.

7.3.2 Experimental Results

The average log-likelihoods for each of the algorithms is shown in Table 7.5. Bold values indicate that the associated algorithms log-likelihood average is statistically significantly better than the other algorithms for the data set.

For all data-sets, OSI-MB performed statistically significantly better than EM and PSO in terms of log-likelihood of the networks found. For the Vote and Car data-sets, OSI-MB significantly outperformed OSI-F, while these two algorithms tied statistically for the Spect and TicTacToe datasets. OSI-F and PSO performed statistically significantly worse than all other algorithms for the Car data-set but outperformed

Table 7.5: Average log-likelihoods for latent variable structure learning

	OSI-MB	OSI-F	EM	PSO
Vote	**-2058.4**	-2344.1	-3356.1	-2359.4
Car	**-3077.9**	-4313.7	-3946.8	-3990.3
Spect	**-1207.1**	**-1240.1**	-2516.7	-1528.4
TicTacToe	**-2447.6**	**-2773.4**	-3900.2	-3391.8

EM for all other data-sets. OSI-F tied statistically with PSO for the Vote and Car data-sets, but OSI-F outperformed PSO for the Spect and TicTacToe data-sets.

7.3.3 Discussion

The paired t-tests on the average log-likelihoods indicate that OSI-MB performed better than the other methods for all data-sets. We believe that the increased performance of OSI-MB is due to each swarm being associated with the Markov blanket of a single node and the resulting communication and competition between overlapping swarms.

The importance of the Markov blanket overlap structure is emphasized by the comparison of OSI-MB and OSI-F. Not only does OSI-MB outperform OSI-F for two of the four networks, but for the Car network, OSI-F was not even competitive with structural EM. It appears that choosing a good overlap structure is crucial to OSI performance for some problems. However, OSI-F still manages to tie with OSI-MB for two of the four networks. This advantage is most likely due to the benefit of splitting the swarms over the nodes of the network combined with the additional exploration obtained by having multiple swarms explore overlapping sections of the search space.

CHAPTER 8

SUMMARY AND CONCLUSIONS

8.1 Summary of Contributions

We have presented several approximation algorithms for the problems of structure learning, parameter estimation, and abductive inference in Bayesian networks. These algorithms are based on a multi-swarm approach to optimization called Overlapping Swarm Intelligence, in which a particle swarm optimizer is split into multiple swarms and a swarm is assigned to each node in the network. We compared these algorithms to existing approaches for the problems of learning and inference in Bayesian networks, in terms of both solution quality and computational complexity.

This work makes several contributions towards improving the quality of algorithms in the areas of structure learning, parameter estimation, and abductive inference in Bayesian networks:

- First, our approach to abductive inference provides a more effective technique for discovering the most probable state assignments for nodes in a Bayesian network given evidence; a problem which is relevant to a number of fields such as fault diagnostics and prognostics. We presented both distributed and non-distributed versions of OSI for this problem and we empirically demonstrated that, while OSI is more computationally expensive than competing methods, both OSI and DOSI significantly outperform the other approaches on most networks studied in terms of the log likelihood of solutions found.

- Second, our approach to parameter estimation will allow for more effective use of latent variable models and will allow for better handling of datasets with

missing data. Our empirical analysis of OSI indicates that OSI outperforms traditional approaches such as EM in terms of 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.

- Third, by developing more accurate techniques for Bayesian structure learning, we will allow automated structure learning to be applied to areas where current techniques are either too computationally expensive or too inaccurate. We presented algorithms for both generative and discriminative structure learning and empirically demonstrated that OSI often outperforms the competing approaches to these problems. We also developed an OSI algorithm for structure learning in the presence of latent variables and demonstrated the effectiveness of this algorithm relative to both structural EM and traditional PSO.

This work also makes a number of contributions in the area of multi-population search algorithms. Specifically dealing with the properties of OSI as a general search technique.

- First, our results on all of the problems that we explored seem to indicate that OSI is an effective search technique for a wide range of problems in both discrete and continuous search spaces. It was found that OSI often outperformed single-swarm PSO, while PSO never significantly outperformed OSI.

- Second, we presented a distributed alternative to OSI that does not require a global network for fitness evaluation. We proved that the number of iterations required for two swarms to reach consensus in DOSI is equal to the length of the minimum path between their nodes, and we empirically demonstrated that DOSI often performs as well as OSI on the abductive inference problem.

- Third, we examined the effect of the swarm architecture and degree of overlap on the performance of OSI. From our experiments on parameter estimation, it appears that, while the amount of overlap does seem to have some effect on performance, the effect of overlap on the performance of OSI may vary between networks. Our experiments comparing different swarm architectures for the abductive inference problem indicate that the overlap structure of the swarms plays a role in algorithm performance. Furthermore, we observed that the Markov blanket architecture achieved better log likelihood than the random architecture for most networks and the no alternative architecture ever outperformed the Markov blanket architecture. These results were reinforced by our experiments on structure learning with latent variables, where, for half of the data-sets, OSI only outperformed the competing approaches when the Markov blanket was used to define the overlap.

- Fourth, we demonstrated that OSI may perform well even when there is little overlap between the swarms. This is supported by our comparison of OSI to other methods for the problem of parameter estimation where we found that OSI outperformed competing approaches, even when there was little overlap between the Markov blankets of latent variables. This is also supported by our results on generative structure learning where, despite the decomposability of the scoring function, the overlap and competition between swarms seemed to give OSI an advantage for most of the data-sets.

The results presented here support the three hypotheses proposed at the beginning of this work. Our first hypothesis was that OSI would outperform single-swarm PSO for most experiments and would never perform worse than single-swarm PSO. This hypothesis is supported by our results for full and partial abductive inference, where

we found that the the state assignments found by OSI and DOSI had higher log-likelihood than those found by DMVPSO for most experiments on complex networks. This hypothesis was further reinforced by our results for parameter estimation, where the likelihood of the data given the model was higher for parameters learned by OSI as compared to those learned by single-swarm PSO.

Additionally, the first hypothesis was supported by our results for Bayesian classifier learning, where the models learned by variants of OSI had higher classification accuracy than those learned by DMVPSO for most datasets, when classification accuracy and conditional mutual information was used to evaluate particle fitness. However, for many of the datasets, OSI and DMVPSO tied statstically when log-likelihood and BIC were used for fitness evaluation. The inability of OSI to outperform single-swarm PSO for these cases is likely due to the poor performance of the log-likelihood and BIC fitness scores when learning Bayesian classifiers.

The first hypothesis was further supported by our results for generative structure learning, both with and without latent variables. For these problems OSI outperformed single-swarm PSO for all data sets, when a good overlap architecture was chosen. Additionally, even when a poor overlap structure was chosen, OSI either outperformed or tied with traditional PSO for all datasets.

For our second hypothesis, we predicted that OSI and DOSI would tie statistically in terms of log-likelihood for most experiments. While this was true for both full and partial abductive inference, there were several experiments in which OSI outperformed DOSI. This may be because DOSI had not reached consensus for these experiments, but more work will be needed to quantify the degree of consensus necessary to achieve good performance.

Our third hypothesis was that the Markov blanket overlap structure would never be outperformed by any other overlap architecture, and would often outperform alter-

native architectures. This was supported by our experiments for both full abductive inference and structure learning.

For full abductive inference, the average log-likelihood for state assignments found by OSI when using the alternative overlap architectures was never significantly greater than the average log-likelihood for state assignments found by OSI when using the Markov blanket architectures. Additionally, OSI often found state assignments with higher log-likelihood when using the Markov blanket architecture, when compared to the state assignments found when using the alternative architectures.

The third hypothesis was further supported by our results for structure learning with latent variables, where OSI often learned models with higher log-likelihood when using the dynamic Markov blanket architecture, as compared to the models learned by the static family-based architecture. However, there were several instances for both full abductive inference, and structure learning with latent variables, where the Markov blanket architecture tied with one or more of the competing architectures. This indicates that an optimal overlap structure may not be necessary for OSI to achieve good performance.

The results presented here demonstrate that OSI is a powerful method that has a clear advantage over single-swarm PSO when applied to a variety of problems. There are many advantages to OSI that may explain why this method outperforms single-swarm PSO on so many problems. First, the representation of each swarm being based on the conditional dependencies present in the underlying problem ensures that each swarm learns the relevant positions for all variables on which the sub-problem depends. Second, since the sub-swarms maintain independence, each swarm can explore a different region of the search space. Third, by splitting the swarms over the nodes of the network, we reduce the possibility of neglecting a potentially good sub-solution for a specific component of the global solution. The above contributions

mark the first steps toward understanding and improving the Overlapping Swarm Intelligence algorithm.

8.2 Future Work

While this work provides an initial exploration of OSI with respect to learning and inference in Bayesian networks, there is much research that may still be done to develop OSI further. There is still much that we do not understand about the algorithm, and there are several ways in which the algorithm could be improved.

One avenue of future research is the exploration of alternative representations and competition strategies to reduce the computational complexity of OSI. For example, it may be possible to vary the number of particles for each swarm based on the complexity of the swarm's Markov blanket without significantly affecting solution quality. In this way, the computational complexity of our algorithm could be reduced by assigning smaller swarms to nodes whose Markov blankets have fewer parameters. Additionally, the algorithm could assign swarms to only a subset of the nodes in the network. Each swarm could optimize over the Markov blanket of its assigned node for some number of iterations t, and after t iterations have elapsed the swarm could be assigned to a new node.

The theory behind OSI needs to be explored further. It may be possible to prove the convergence of OSI under certain assumptions, as was done with single-swarm PSO by Van den Bergh *et al.* [86]. Drawing from the approaches of [57] and [58] on the distributed consensus problem to analyze the relationship between OSI convergence rate and the structure of the sub-swarms is one possible avenue for this research.

Additionally, research must be done to examine the effect of overlap on the performance of OSI. We conjecture that, for many optimization problems, optimal overlap

structures are related to the conditional dependence properties of the underlying fitness landscape. Further work is needed to verify the existence of optimal swarm overlap structures, and methods must be developed to identify good overlap structures for OSI when applied to a given problem. This problem could be addressed by learning a graphical model from the data associated with a given problem whereby we could determine each variable's Markov blanket. This would then allow OSI to be applied to a larger spectrum of optimization problems.

Also, more research must be done to extend the theory behind DOSI. This would include evaluating the effect of consensus on the performance of DOSI. It is possible that full consensus is not necessary to achieve good solution quality. If this is true, it would increase the chance of distributing these algorithms efficiently. Another avenue for future research is the exploration of additional techniques for achieving consensus. For instance, instead of certain nodes acting as an arbiter, propagating correct scores throughout the network, nodes could propagate personal best scores that would then be included in the personal solutions of other nodes through a fitness-based weighted average.

The effectiveness of OSI on the problems presented in this work, indicates that similar distributed search techniques may be combined effectively with population-based algorithms such as evolution strategies, genetic algorithms, or differential evolution. It could be that the techniques used by OSI are even applicable to local and stochastic search techniques such as hill-climbing and simulated annealing. Further research must be done to verify this intuition and establish a general framework for distributed search.

Finally, OSI must be compared against more recent state-of-the-art techniques for Bayesian network structure learning, both with complete data and with latent

variables. While the results presented here are encouraging, more experiments must be performed to demonstrate the competitiveness of OSI in this space.

8.3 Publications

Several papers have been published, which directly relate to the research presented here:

- Nathan Fortier, John Sheppard, Karthik Pillai. Bayesian abductive inference using overlapping swarm intelligence. *IEEE Swarm Intelligence Symposium (SIS)*, pages 263–270, 2013
- Nathan Fortier, John Sheppard, Shane Strasser. Abductive inference in Bayesian networks using distributed overlapping swarm intelligence. *Soft Computing*, pages 1–21, 2014
- Nathan Fortier, John Sheppard, Shane Strasser. Learning Bayesian classifiers using overlapping swarm intelligence. *IEEE Symposium on Swarm Intelligence (SIS)*, pages 1–8. IEEE, 2014
- Nathan Fortier, John Sheppard, Shane Strasser. Parameter estimation in Bayesian networks using overlapping swarm intelligence. *To appear in the Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2015)*, 2015

REFERENCES CITED

[1] Min Zou, Suzanne D Conzen. A new dynamic Bayesian network (dbn) approach for identifying gene regulatory networks from time course microarray data. *Bioinformatics*, 21(1):71–79, 2005.

[2] Gennady Agre. Diagnostic Bayesian networks. *Computers and Artificial Intelligence*, 16(1), 1996.

[3] Uri Lerner, Ronald Parr, Daphne Koller, Gautam Biswas, i in. Bayesian fault detection and diagnosis in dynamic systems. *AAAI/IAAI*, pages 531–537, 2000.

[4] Gernot D Kleiter. Bayesian diagnosis in expert systems. *Artificial Intelligence*, 54(1):1–32, 1992.

[5] Daniel Nikovski. Constructing Bayesian networks for medical diagnosis from incomplete and partially correct statistics. *IEEE Transactions on Knowledge and Data Engineering*, 12(4):509–516, 2000.

[6] Igor Kononenko. Inductive and Bayesian learning in medical diagnosis. *Applied Artificial Intelligence*, 7(4):317–337, 1993.

[7] Patrick Joseph Donnelly. Bayesian approaches to musical instrument classification using timbre segmentation. *Dissertation Proposal*. Montana State University, Department of Computer Science.

[8] David Maxwell Chickering. Learning Bayesian networks is NP-complete. *Learning from data*, pages 121–130. Springer, 1996.

[9] Tong Wang, Jie Yang. A heuristic method for learning Bayesian networks using discrete particle swarm optimization. *Knowledge and information systems*, 24(2):269–281, 2010.

[10] Olivier Regnier-Coudert, John McCall. An island model genetic algorithm for Bayesian network structure learning. *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.

[11] Yanghui Wu, John McCall, David Corne. Two novel ant colony optimization approaches for Bayesian network structure learning. *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–7. IEEE, 2010.

[12] Daphne Koller, Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[13] S.E. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68:399–410, 1994.

[14] Paul Dagum, Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993.

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

[16] Robert Fung, Brendan Del Favero. Applying bayesian networks to information retrieval. *Communications of the ACM*, 38(3):42–ff, 1995.

[17] Paolo Trucco, Enrico Cagno, Fabrizio Ruggeri, Ottavio Grande. A bayesian belief network modelling of organisational factors in risk analysis: A case study in maritime transportation. *Reliability Engineering & System Safety*, 93(6):845–856, 2008.

[18] Carmel A Pollino, Owen Woodberry, Ann Nicholson, Kevin Korb, Barry T Hart. Parameterisation and evaluation of a bayesian network for use in an ecological risk assessment. *Environmental Modelling & Software*, 22(8):1140–1152, 2007.

[19] Philippe Weber, Gabriela Medina-Oliva, Christophe Simon, Benoît Iung. Overview on bayesian networks applications for dependability, risk analysis and maintenance areas. *Engineering Applications of Artificial Intelligence*, 25(4):671–682, 2012.

[20] Min Zou, Suzanne D Conzen. A new dynamic bayesian network (DBN) approach for identifying gene regulatory networks from time course microarray data. *Bioinformatics*, 21(1):71–79, 2005.

[21] Nir Friedman, Michal Linial, Iftach Nachman, Dana Pe'er. Using bayesian networks to analyze expression data. *Journal of computational biology*, 7(3-4):601–620, 2000.

[22] Arunkumar Chinnasamy, Wing-Kin Sung, Ankush Mittal. Protein structure and fold prediction using tree-augmented naive bayesian classifier. *Journal of Bioinformatics and Computational Biology*, 3(04):803–819, 2005.

[23] J Uebersax. Genetic counseling and cancer risk modeling: An application of bayes nets. *Marbella, Spain: Ravenpack International*, 2004. Methodology Division Research Report for RavenPack International.

[24] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.

[25] S.E. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68:399–410, 1994.

[26] James D Park. Map complexity results and approximation methods. *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 388–396. Morgan Kaufmann Publishers Inc., 2002.

[27] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. *Proceedings of the Twelfth international conference on Uncertainty in artificial intelligence*, pages 211–219. Morgan Kaufmann Publishers Inc., 1996.

[28] Gregory F Cooper, Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.

[29] Gideon Schwarz, i in. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[30] Ernst Wit, Edwin van den Heuvel, Jan-Willem Romeijn. All models are wrong: an introduction to model uncertainty. *Statistica Neerlandica*, 66(3):217–236, 2012.

[31] Gregory F Cooper, Edward Herskovits. A Bayesian method for constructing bayesian belief networks from databases. *Proceedings of the Seventh conference on Uncertainty in Artificial Intelligence*, pages 86–94. Morgan Kaufmann Publishers Inc., 1991.

[32] Nir Friedman, Dan Geiger, Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[33] C Chow, C Liu. Approximating discrete probability distributions with dependence trees. *Information Theory, IEEE Transactions on*, 14(3):462–467, 1968.

[34] Stuart J. Russell, Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, edition 2, 2003.

[35] Arthur P Dempster, Nan M Laird, Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[36] Nir Friedman. The Bayesian structural EM algorithm. *Proceedings of the 14th conference on Uncertainty in artificial intelligence*, pages 129–138. Morgan Kaufmann Publishers Inc., 1998.

[37] Russell Eberhart, James Kennedy. A new optimizer using particle swarm theory. *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE, 1995.

[38] Maurice Clerc, James Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[39] Vladimiro Miranda, Hrvoje Keko, Alvaro Jaramillo Duque. Stochastic star communication topology in evolutionary particle swarms (epso). *International Journal of Computational Intelligence Research*, 4(2):105–116, 2008.

[40] Peng-Yeng Yin, Fred Glover, Manuel Laguna, Jia-Xian Zhu. Cyber swarm algorithms–improving particle swarm optimization using adaptive memory strategies. *European Journal of Operational Research*, 201(2):377–389, 2010.

[41] Wesam Elshamy, Hassan M Emara, Ahmed Bahgat. Clubs-based particle swarm optimization. *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 289–296. IEEE, 2007.

[42] Rui Mendes. *Population topologies and their influence in particle swarm performance*. Doctoral Dissertation, Citeseer, 2004.

[43] J. Kennedy, R.C. Eberhart. A discrete binary version of the particle swarm algorithm. *Systems, Man, and Cybernetics, 1997. IEEE International Conference on Computational Cybernetics and Simulation.*, Volume 5, pages 4104–4108, oct 1997.

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

[45] Nathan Fortier, John Sheppard, Karthik Pillai. Bayesian abductive inference using overlapping swarm intelligence. *IEEE Swarm Intelligence Symposium (SIS)*, pages 263–270, 2013.

[46] Nathan Fortier, John Sheppard, Shane Strasser. Abductive inference in Bayesian networks using distributed overlapping swarm intelligence. *Soft Computing*, pages 1–21, 2014.

[47] R. Tanese, J.H. Co-Chairman-Holland, Q.F. Co-Chairman-Stout. Distributed genetic algorithms for function optimization. *Proceedings of the International Conference on Genetic Algorithms*, pages 434–439. University of Michigan, 1989.

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

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

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

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

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

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

[54] Nathan Fortier, John W Sheppard, Karthik Ganesan Pillai. DOSI: Training artificial neural networks using overlapping swarm intelligence with local credit assignment. *Joint 6th International Conference on Soft Computing and Intelligent Systems (SCIS) and 13th International Symposium on Advanced Intelligent Systems (ISIS)*, pages 1420–1425. IEEE, 2012.

[55] M. Rabbat, R. Nowak. Distributed optimization in sensor networks. *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 20–27. ACM, 2004.

[56] Reza Olfati-Saber, J Alex Fax, Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

[57] S. Patterson, B. Bamieh, A. El Abbadi. Convergence rates of distributed average consensus with stochastic link failures. *IEEE Transactions on Automatic Control*, 55(4):880–892, 2010.

[58] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[59] Rina Dechter. Mini-buckets: A general scheme for generating approximations in automated reasoning. *IJCAI*, Volume 97, pages 1297–1303. Citeseer, 1997.

[60] D. Nilsson. An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, 8(2):159–173, 1998.

[61] A.P. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2(1):25–36, 1992.

[62] L.M. de Campos, J.A. Gamez, S. Moral. Partial abductive inference in Bayesian belief networks by simulated annealing. *International Journal of Approximate Reasoning*, 27(3):263 – 283, 2001.

[63] K. Kask, R. Dechter. Stochastic local search for Bayesian networks. *Workshop on AI and Statistics*, pages 113–122. Morgan Kaufman Publishers, 1999.

[64] Elvira Consortium. Elvira: An environment for creating and using probabilistic graphical models. *Proceedings of the first European workshop on probabilistic graphical models*, pages 222–230, 2002.

[65] E.S Gelsema. Abductive reasoning in Bayesian belief networks using a genetic algorithm. *Pattern Recognition Letters*, 16:865–871, 1995.

[66] Carlos Rojas-Guzman, Mark A Kramer. Galgo: A genetic algorithm decision support tool for complex uncertain systems modeled with Bayesian belief networks. *Proceedings of the Ninth international conference on Uncertainty in artificial intelligence*, pages 368–375. Morgan Kaufmann Publishers Inc., 1993.

[67] L.M. de Campos, J.A. Gamez, S. Moral. Partial abductive inference in Bayesian belief networks using a genetic algorithm. *Pattern Recognition Letters*, 20:1211–1217, 1999.

[68] N. Sriwachirawat, S. Auwatanamongkol. On approximating k-MPE of Bayesian networks using genetic algorithm. *Cybernetics and Intelligent Systems*, pages 1–6, 2006.

[69] K. Ganesan Pillai, J. W. Sheppard. Abductive inference in Bayesian belief networks using swarm intelligence. *Joint 6th International Conference on Soft Computing and Intelligent Systems (SCIS) and 13th International Symposium on Advanced Intelligent Systems (ISIS)*, pages 375–380, 2012.

[70] Greg CG Wei, Martin 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.

[71] Gilles Celeux, Jean 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.

[72] Gal Elidan, Matan Ninio, Nir Friedman, Dale Shuurmans. Data perturbation for escaping local maxima in learning. *Proceedings of the National Conference on Artificial Intelligence*, pages 132–139, 2002.

[73] Gal Elidan, Nir Friedman. Learning hidden variable networks: The information bottleneck approach. *Journal of Machine Learning Research*, pages 81–127, 2005.

[74] Naftali Tishby, Fernando C Pereira, William Bialek. The information bottleneck method. *In Proc. of the 37th Annual Allerton Conference on Communication, Control and Computing*, page 368377, 1999.

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

[76] Ole J Mengshoel, Avneesh Saluja, Priya Sundararajan. Age-layered expectation maximization for parameter learning in Bayesian networks. *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, pages 984–992, 2012.

[77] Changhe Yuan, Brandon Malone. Learning optimal Bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research*, 48:23–65, 2013.

[78] David Heckerman. *A tutorial on learning with Bayesian networks*. Springer, 1998.

[79] Marco Scutari. Bayesian network repository, 2012. `http://www.bnlearn.com/bnrepository/`.

[80] John Binder, Daphne Koller, Stuart Russell, Keiji Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29(2-3):213–244, 1997.

[81] Bruce Abramson, John Brown, Ward Edwards, Allan Murphy, Robert L Winkler. Hailfinder: A Bayesian system for forecasting severe weather. *International Journal of Forecasting*, 12(1):57–71, 1996.

[82] Russ C Eberhart, Yuhui Shi. Comparing inertia weights and constriction factors in particle swarm optimization. *Proceedings of the Congress on Evolutionary Computation*, Volume 1, pages 84–88. IEEE, 2000.

[83] Pat Langley. Machine learning as an experimental science. *Machine Learning*, 3(1):5–8, 1988.

[84] Richard E Neapolitan. *Learning Bayesian networks*. Pearson Prentice Hall Upper Saddle River, 2004.

[85] D.J. Newman A. Asuncion. UCI machine learning repository, 2007. `https://archive.ics.uci.edu/ml/datasets.html`.

[86] Frans van den Bergh, AP Engelbrecht. A new locally convergent particle swarm optimizer. *Proceedings of the IEEE international conference on systems, man, and cybernetics*, Volume 7, pages 6–9, 2002.

[87] Nathan Fortier, John Sheppard, Shane Strasser. Learning Bayesian classifiers using overlapping swarm intelligence. *IEEE Symposium on Swarm Intelligence (SIS)*, pages 1–8. IEEE, 2014.

[88] Nathan Fortier, John Sheppard, Shane Strasser. Parameter estimation in Bayesian networks using overlapping swarm intelligence. *To appear in the Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2015)*, 2015.