# Bayesian Abductive Inference using Overlapping Swarm Intelligence

Nathan Fortier
Department of Computer Science
Montana State University
Bozeman MT, 59717-3880
Email: nathan.fortier@msu.montana.edu

John Sheppard
Department of Computer Science
Montana State University
Bozeman MT, 59717-3880
Email: john.sheppard@cs.montana.edu

Karthik Ganesan Pillai
Department of Computer Science
Montana State University
Bozeman MT, 59717-3880
Email: k.ganesanpillai@cs.montana.edu

***Abstract*—Abductive inference in Bayesian networks, is the problem of finding the most likely joint assignment to all non-evidence variables in the network. Such an assignment is called the most probable explanation (MPE). A novel swarm-based algorithm is proposed that finds the $k$-MPE of a Bayesian network. Our approach is an overlapping swarm intelligence algorithm in which a particle swarm is assigned to each node in the network. Each swarm searches for value assignments for its node's Markov blanket. Swarms that have overlapping value assignments compete to determine which assignment will be used in the final solution. In this paper we compare our algorithm to several other local search algorithms and show that our approach outperforms the competing methods in its ability to find the $k$-MPE.**

## I. Introduction

Bayesian networks are directed acyclic graphs in which nodes represent random variables and edges represent conditional dependencies. Each node in the network contains a probability function that takes as input a set of values for the node's parent variables and returns the probability distribution of the variable represented by the node.

While Bayesian networks are a useful model for probabilistic reasoning under uncertainty, one of the important problems in using Bayesian networks is finding the maximum *a posteriori* probability state of the network given the evidence. This problem is known as abductive inference. If we let $M = X \backslash E$, the task of abductive inference is to find the most likely assignment to the variables in $M$ given the evidence $E = e$:

$$MPE(M, e) = \underset{m \in M}{\operatorname{argmax}}\, p(m|e)$$

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 [1], 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 [2] it was shown that even problem of finding a constant factor approximation of the $k$-MPE is NP-hard.

In this paper, we introduce a swarm based approximation algorithm to solve the abductive inference problem using overlapping swarm intelligence (OSI), first introduced by Haberman and Sheppard [3]. In our approach a particle swarm is associated with each non-evidence node in the network. Each swarm learns the value assignments for the variables in the Markov blanket associated with that swarm's node. Swarms that learn value assignments for the same variable are said to overlap. Such swarms will compete to determine which value assignment should be used in the final solution. Each swarm in our approach uses the discrete multi-valued PSO algorithm proposed in [4] to search for partial solutions. We will compare the results of our approach to several other local search algorithms including the PSO-based algorithm proposed by Ganesan Pillai and Sheppard in [5]. We hypothesize that our algorithm will outperform the algorithm proposed in [5] in terms of the log likelihood of $k$-MPE found when used to perform inference on complex networks. We also compare our results to a hillclimbing-based algorithm [6] as a baseline and further hypothesize we will outperform that algorithm on the same measure.

This paper is organized as follows. In section II we provide background on Bayesian belief networks and Particle Swarm Optimization. Next, we provide a review of related literature. In section IV we present our approach to solve the abductive inference problem using OSI. We describe our experimental design in section V. In section VI we present the results of our experiments. Next, we discuss the implications of our experimental results. We then present our conclusions and discuss possible future work in section VII.

## II. Background

### A. Bayesian Networks

A Bayesian network is a directed acyclic graph that represents a joint probability distribution in which the nodes are a set of random variables which can assume an arbitrary number of mutually exclusive values [7]. Edges between nodes in the network represent a probabilistic relationships between the nodes. Each root node contains a set of prior probabilities while each non-root node contains a set of conditional probabilities conditioned on the node's parents. For any set of random variables in the network, the probability of any entry

978-1-4673-6004-3/13/$31.00 

of the joint distribution can be computed using the chain rule.

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

The structure of the network allows conditional independence relationships to be described by the Bayesian network, thus enabling the distribution to be represented as

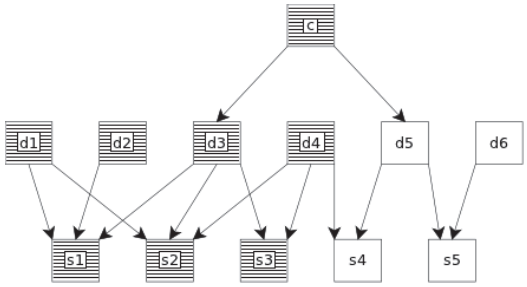
$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | \text{Pa}(X_i)).$$

We say a variable $X_i$ is conditionally independent of all other variables in the network given its Markov blanket, which consists of the a node's parents, children, and childrens parents.

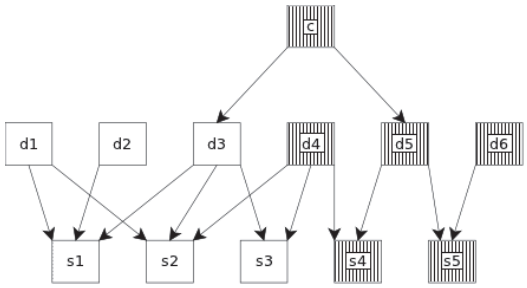
$$\{X_i \perp (\mathbf{X} \backslash (\{X_i\} \cup MB(X_i))) \, | MB(X_i)\}$$

An example illustrating the concept of a Markov blanket is shown in Figure 1. Figure 1a shows the Markov blanket of $d_3$, Figure 1b shows the Markov blanket of $d_5$, and Figure 1c shows the Markov blankets of both $d_3$ and $d_5$. In the example, nodes in the Markov blanket of $d_3$ are shown with a horizontal hash pattern and nodes in the Markov blanket of $d_5$ are shown with a vertical hash pattern. In Figure 1c nodes that are in the Markov blankets of both $d_3$ and $d_5$ (namely $c$ and $d_4$) show both horizontal and vertical hashes, thus indicating an overlap. We will exploit these overlaps later.

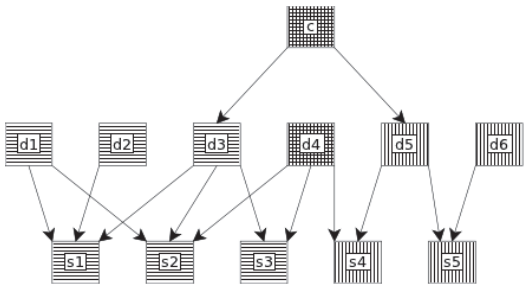
There are several problems that can be solved with Bayesian networks. One such problem is that of finding the posterior probability of a random variable given a set of evidence $E$. The abductive inference problem extends this idea to find the $k$ most probable variable assignments given the set of evidence $E = e$. Both inference problems are known to be NP-hard [8],[1] and, in fact, the abductive inference problem has been shown to be NP-hard to approximate within a constant factor [2].



(a) Markov blanket of $d_3$



(b) Markov blanket of $d_5$



(c) Overlap of $d_3$ and $d_5$

Fig. 1: Markov blanket example

### *B. Particle Swarm Optimization*

The Particle Swarm Optimization (PSO) algorithm is a search technique based on the social behavior of fish schools and bird flocks first proposed by Eberhart and Kennedy [9]. PSO is a population-based search technique in which the population is initialized with random solution vectors called particles. During the search process the positions of each particle are updated based on that particle's corresponding velocity vector. A particle's velocity vector is updated based on the fitness of the states visited by that particle. Through this process the particles move closer to an optimum in the search space. The pseudocode for the traditional PSO algorithm is presented in Algorithm 1.

The algorithm begins by randomly initializing a swarm of particles over the search space. At each iteration of the algorithm the fitness of a particle, $x_i$, is calculated using the fitness function, $f(x_i)$. The personal best position for that particle is stored in the vector $p_i$. The global best position found among all particles is stored 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. This is an important aspect of the algorithm that provides a balance between exploration and exploitation.

In Algorithm 1, **P** is the particle swarm, $\text{U}(0, \phi_i)$ is a vector of random numbers uniformly distributed in the interval $[0, \phi_i]$, $\otimes$ is component-wise multiplication, $v_i$ is the velocity of a particle and $x_i$ is the position of a particle.

Three parameters need to be defined for the PSO algorithm:

- $\phi_1$ determines the maximum force with which a particle is pulled toward $p_i$;
- $\phi_2$ determines the maximum force with which a particle is pulled toward $p_g$;
- $\omega$ is the inertia weight.

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

**Algorithm 1** Particle Swarm Optimization

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

### C. Discrete Particle Swarm Optimization

Kennedy and Eberhart proposed a modification of the traditional PSO algorithm for problems with binary-valued solution elements [10]. In this algorithm, each particle's position is a vector from the $d$-dimensional binary solution space $x_i \in \{0, 1\}^d$ and each particle's velocity is a vector from the $d$-dimensional continuous space, $v_i \in \Re^d$. Each velocity term denotes the probability of a particle's position term having a value of 0 or 1 in the next iteration. Each particle's velocity is updated as described in [9] while each particle's position is updated using the following equation:

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

While this algorithm has been shown to be effective, it is limited to discrete problems with binary valued solution elements.

To relax the binary state assumption Veeramachaneni *et al.* propose a discrete multi-valued PSO (DMVPSO) algorithm [4]. 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. Each particle's velocity is a $d$-dimensional vector of continuous values. The velocity is transformed into a number between $[0, M]$ using the following equation:

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

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

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

## III. Related work

### A. Traditional Approaches to the $k$-MPE Problem

Dechter et al. [11] proposed an exact algorithm to solve the k-MPE problem called bucket elimination. The algorithm uses a variable elimination process in which the node with the fewest neighbors in eliminated at each iteration. Bucket elimination uses max-marginalization instead of sum-marginalization when eliminating a variable and the most probable state assignment for the variable is stored. Like variable elimination, this algorithm has exponential time complexity.

Nilsson et al. [12] describes an divide and conquer algorithm that provides an exact solution to the $k$-MPE problem. This algorithm is based on Dawid's flow propagation algorithm [13] for calculating the $k$-MPE for junction trees. While the algorithm is faster than other exact abductive inference algorithms such as bucket elimination, it has exponential time complexity and is unfeasible for large networks.

Kask and Dechter proposed a stochastic local search algorithm for solving the MPE problem [6]. In their approach, a hillclimbing algorithm was combined with Gibbs Sampling. The results of the author's experiments indicate that their approach outperforms other techniques such as stochastic simulation, simulated annealing, or hillclimbing alone.

In the Elvira [14] software environment a solution to the $k$-MPE Problem is approximated using a junction tree based algorithm. This algorithm is based on Nilssons algorithm but approximate probability trees are used in place of the true probability trees.

### B. Soft Approaches to the $k$-MPE Problem

Several soft computing techniques have been used to to find approximate solutions to the $k$-MPE problem. In [15] several computational experiments are described that use genetic algorithms for abductive inference in Bayesian networks. In their approach, the states of the variables in the Bayesian network are represented by a chromosome corresponding to a string of integers between 0 and 1. 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. Thus, to calculate a chromosome's fitness, $|M|$ multiplications are needed. This fitness function is effective since $p(M|e) \propto p(M, e)$.

Rojas-Guzman *et al.* proposed a graph-based evolutionary algorithm for performing approximate abductive inference on Bayesian Networks[16]. Here the authors present a genetic algorithm in which each chromosome is represented by a graph. Each chromosome specifies 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 set of assignments.

Partial abductive inference is the task of finding $k$-MPE for a subset of the variables in the network. An approach for performing approximate partial abductive inference using a

genetic algorithm was proposed by Campos *et al.* [17]. In this approach, the state assignments for the subset of variables are represented as a chromosome consisting of integers. Each position in the chromosome represents the state assignment for the corresponding variable. To evaluate the fitness of each chromosome, probabilistic propagation is used.

Sriwachirawat *et al.* proposed a niching genetic algorithm (NGA) designed to utilize the "multifractal characteristic and clustering property" of Bayesian networks to find $k$-MPE [18]. This algorithm is based on a niching method and it 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 in [15].

In [5], a discrete multi-valued PSO (DMVPSO) approach for finding $k$-MPE is proposed. This approach uses the algorithm described in [4] to search for probable state assignments. In this algorithm, each particle's set of object parameters is represented by a string of integers. Each integer corresponds to a state assignment for a 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 used in [15] and [18].

### C. Distributed Optimization

Much work has been done in the area of distributed optimization. Patterson et al. [19] analyzed the convergence rate of the distributed average consensus algorithm. This work also includes an analysis of the relationship between the convergence rate and the network topology.

Boyed et al. [20] discusses convex distributed optimization in the context of statistics and machine learning. The authors argue 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 which are solved and used to find a solution to a large global problem. The authors show that this approach can be applied to a wide variety of distributed optimization problems.

Rabbat et al. [21] analyzed the convergence of distributed optimization algorithms in sensor networks. The authors prove that for a large set of problems such algorithms converge to a solution within a certain distance of the global optimum.

### D. Distributed Soft Computing

Several modifications to traditional soft computing methods have been proposed in which the single large population is replaced by smaller distributed subpopulations. We refer to these methods as distributed soft computing algorithms.

Several authors have discussed distributed genetic algorithms (GA) which are commonly reffered to as Island Models [22], [23], [24], [25]. In these models several subpopulations known as islands are maintained by the genetic algorithm and members of the populations are exchanged through a process called migration. These methods have been shown to obtain better quality solutions than traditional GAs [23]. Because each of 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 [25].

Bergh and Engelbrecht [26] proposed several distributed PSO algorithms for the training of feedforward neural networks. These methods include NSPLIT in which there is a single particle swarm for each neuron in the network and LSPLIT in which there is a swarm assigned to each layer of the network. The results obtained by Bergh and Engelbrecht indicate that the distributed algorithm outperforms traditional PSO methods.

Recently another distributed approach to improve the performance of the PSO algorithm has been explored in which multiple swarms are assigned to overlapping subproblems. This approach is called Overlapping Swarm Intelligence (OSI) [3], [27], [28]. 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.

In 2010, Haberman and Sheppard [3] used the OSI method to develop an energy-efficient routing protocol for sensor networks that ensures reliable path selection while minimizing the energy consumption for the route selection process. Their 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 [27] developed an OSI algorithm for training the weights of deep artificial neural networks. In their approach, the structure of the network is separated 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 which describes 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 and the traditional PSO algorithm. A distributed version of this approach was developed subsequently by Fortier, Sheppard, and Ganesan Pillai [28].

## IV. Approach

Here we describe an approach to approximate $k$-MPE based on the PSO algorithm proposed in [5] but focusing on learning sub-problems similar to the approaches in [3] and [27]. In our approach we associate a swarm with each node in the network. A node's corresponding swarm learns the state assignments associated with that node's Markov blanket. This representation is advantageous since every node in the network is conditionally independent of a node $X$ when conditioned

on its Markov blanket. The pseudocode for our approach is presented in Algorithm 2.

A set of global state assignments $\alpha$ is maintained across all swarms and is used for inter-swarm communication. The set $\alpha$ initially contains only one state assignment obtained through a forward sampling process in which the nodes of the network are assigned a state following a topological ordering of the network. In forward sampling, the probability of picking a given state is determined by the node's distribution and the sampled state of the node's parents. Each particle's set of object parameters consists of a vector of integers. Then each integer corresponds to the state of a variable in the swarm's Markov blanket. This means that each particle represents a partial state assignment for the network. We determine the quality of a complete state assignment as follows:

$$\begin{aligned} q(X) &= \log\left(\prod_{X_i \in X} P(X_i|\text{Pa}(X_i))\right) \\ &= \sum_{X_i \in X} \log P(X_i|\text{Pa}(X_i)). \end{aligned}$$

where $X = \{X_1, X_2 ... X_n\}$ is a complete state assignment and $\text{Pa}(X_i)$ corresponds to the assignments for the parents of $X_i$.

Given a partial state assignment $x_p$ represented by some particle $p$ in swarm $s$ and the set of complete global state assignments $\alpha = \{A_1, \ldots, A_k\}$ we can construct a new set of state assignments $\beta_p = \{B_1, \ldots, B_k\}$ by inserting $x_p$ into each state assignment $A_i \in \alpha$ as follows:

$$\forall B_i \in \beta_p \quad B_i = x_p \cup \{A_i \backslash mb_s\}$$

where $mb_s$ consists of the state assignments for the Markov blanket of swarm $s$ within $A_i$. We use $\beta_p$ to calculate the fitness of each particle,

$$f(p) = \sum_{B_i \in \beta_p} q(B_i)$$

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

At the end of each iteration of the algorithm, swarms that share a node in the network (such as $c$ and $d_4$ in Figure 1) will compete to determine which state is assigned to the node in each assignment $A_m \in \alpha$. This competition is held between the state assignments found by the personal best particles in each swarm. The state that results in the highest log-likelihood is the one selected for inclusion. This process is shown in Algorithm 3. In the example presented in Figure 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 $H$. At each iteration the $k$ most probable assignments in $H$ are added to $\alpha$. Once the algorithm has terminated $\alpha$ is returned.

**Algorithm 2** Overlapping Swarm Intelligence

```
Initialize α using forward sampling
Initialize particles in each swarm
Create an empty list of assignments H

repeat
  for each swarm s do
    for each particle, p ∈ s do
      Construct β_p
      Add β_p to H
      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

  α ← k most probable assignments in H

  for each state assignment A ∈ α do
    for each node n in the network do
      Let S be all swarms where n ∈ mb_s
      Let v_n be the state of n in A
      v_n ← compete(S, n, A)
    end for
  end for
  Add α to H
until termination criterion is met

return α
```

**Algorithm 3** $compete(S, n, A)$

```
bestQ ← −∞
for each swarm s ∈ S do
  Let p_g be the most fit particle in s
  Let v_n be the state of n within p_g
  Insert v_n into A
  if q(A) > bestQ then
    bestQ ← q(A)
    bestV ← v_n
  end if
end for
```

## V. Experimental Design

We compare our algorithm to the methods proposed in [5], [6], [15], and [18]. As discussed above, the algorithm proposed in [5] is an adaption of the DMVPSO algorithm to the problem of abductive inference. The algorithm proposed in [6] is a greedy hillclimbing algorithm provided as a baseline for comparison. Finally, the algorithms proposed in [15] and

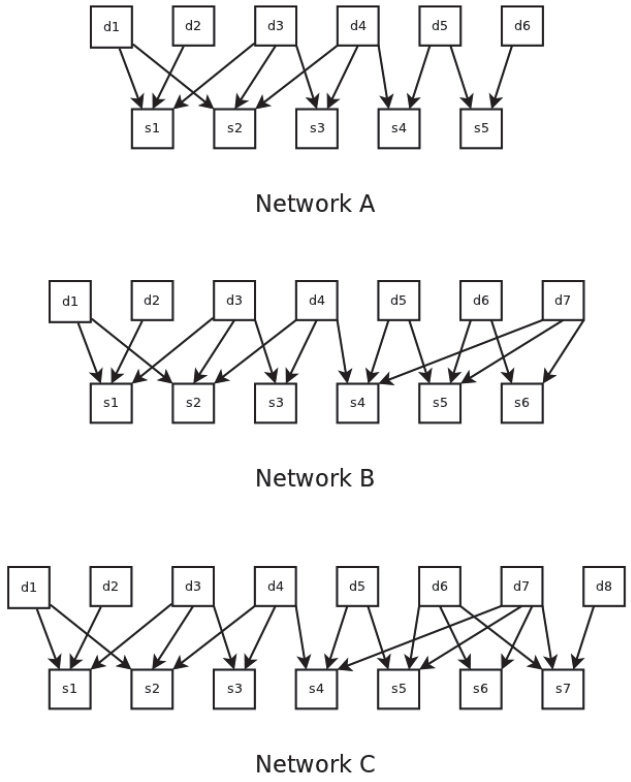



Fig. 2: Bipartite Bayesian networks used for experiments.

[18] are a standard adaptation of the genetic algorithm and a probabilistic restricted mating genetic algorithm (PRMGA), compared in [5]. The latter algorithm adapts a probabilistic crowding procedure to encourage crossover between individuals that belong to the same cluster that contains other high-fitness individuals.

To compare these algorithm we used the bipartite networks presented in Figure 2 along with four additional Bayesian networks obtained from the "bnlearn" Bayesian Network Repository [29]. The networks taken from the Bayesian Network Repository are the Win95pts, Insurance, Hailfinder, and Hepar2 networks. The networks shown in Figure 2 were taken from [5]. For these networks, parameters for root nodes were generated according to their priors while parameters for non-root nodes were generated based on their parents. Each node in the networks presented in Figure 2 has three states. The properties for all networks are shown in Table I. For all networks each leaf node in the network was set as evidence with a $50\%$ probability. The state of each evidence variable was chosen uniformly at random.

For each network, four experiments were performed, based on the number of $k$-MPE. We evaluated all algorithms with $k$ set to 2, 4, 6, and 8 respectively. 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 $\sigma$ was set to 0.2. The value for $\sigma$ was taken from [5] to ensure consistency of results. For the genetic algorithms

TABLE I: Statistics of the various networks

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

the population size was set to 20. All algorithms were run until convergence. The sums of the log likelihoods for the $k$ most fit solutions found in each run were averaged over the ten runs of each algorithm and compared using a paired t-test to evaluate significance. The confidence interval for the t-test is $95\%$. We also measured the number of fitness evaluations performed by each of the algorithms in order to compare the computational complexity of each approach.

The $k$-MPE solutions for the algorithm proposed in [5] are generated using the approach described. For each generation, the $k$ best solutions are stored in a queue. If an individual's solution is not in the queue and its fitness is higher than that of the least-fit solution in the queue, the least-fit solution is replaced by the new individual's solution.

## VI. Results

In Table II we present the log likelihoods obtained from our experiments. Here 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 $k$. Algorithms that tie statistically for best are bolded.

For the networks Win95pts, Insurance, Hailfinder, and Hepar2 we observe that, based on the paired t-tests on log likelihood, the OSI algorithm has the best performance. For Network A all algorithms tie statistically when k is set to 2, NGA and OSI tie statistically for best when $k$ is set equal to 6, and GA and OSI tie statistically for best when $k$ is set equal to 8. For Network B GA and OSI tie statistically for best when $k$ is set equal to 2, 6, and 8. For Network C DMVPSO and OSI tie statistically for best when $k$ is set equal to 2.

In Figure 3 we present a bar graph comparing the number of fitness evaluations required by each of the algorithms with respect to the number of nodes in the network.

## VII. Discussion

The paired t-tests on the sum of the log likelihoods indicate that OSI performed either equal to or better than the other methods for all values of $k$. While the OSI approach does not appear to have an advantage when used on small networks such as Network A and Network B, we can see that it performed better than the other methods for all networks containing more than 15 nodes. This indicates that our approach

TABLE II: Average sum of log likelihoods for different values of $k$

| Network | $k$ | OSI | Greedy | NGA | GA | DMVPSO |
|---|---|---|---|---|---|---|
| Network A | 2 | **-14.52 ± 0.00** | **-17.61 ± 6.19** | **-14.87 ± 0.56** | **-14.74 ± 0.59** | **-14.71 ± 0.34** |
| | 4 | **-32.30 ± 0.00** | -36.37 ± 3.96 | **-33.39 ± 1.86** | **-36.37 ± 1.34** | **-32.30 ± 0.95** |
| | 6 | **-40.30 ± 0.00** | -55.86 ± 7.38 | **-41.22 ± 1.49** | -43.16 ± 2.71 | -42.47 ± 1.37 |
| | 8 | **-68.23 ± 0.00** | -85.27 ± 6.88 | -70.25 ± 2.27 | **-67.76 ± 3.51** | -71.30 ± 1.62 |
| Network B | 2 | **-18.39 ± 0.21** | -24.29 ± 5.08 | -19.94 ± 1.27 | **-19.24 ± 1.18** | -19.79 ± 0.64 |
| | 4 | **-29.18 ± 0.00** | -51.13 ± 8.00 | -33.24 ± 2.16 | -31.34 ± 1.61 | -31.62 ± 1.56 |
| | 6 | **-49.70 ± 0.00** | -67.15 ± 6.34 | **-50.57 ± 2.49** | **-49.85 ± 3.06** | -51.93 ± 2.36 |
| | 8 | **-63.99 ± 0.00** | -96.59 ± 9.58 | -67.56 ± 2.64 | **-62.14 ± 4.88** | -67.64 ± 2.97 |
| Network C | 2 | **-21.68 ± 0.00** | -26.49 ± 3.90 | -22.77 ± 0.93 | -22.76 ± 0.92 | **-22.38 ± 1.04** |
| | 4 | **-32.08 ± 0.55** | -51.93 ± 7.13 | -41.03 ± 2.41 | -37.06 ± 3.34 | -39.47 ± 2.35 |
| | 6 | **-72.36 ± 0.23** | -93.05 ± 8.02 | -85.14 ± 4.80 | -78.10 ± 5.54 | -77.60 ± 2.98 |
| | 8 | **-87.69 ± 0.19** | -110.23 ± 9.65 | -103.60 ± 4.98 | -94.55 ± 4.66 | -96.81 ± 5.97 |
| Win95pts | 2 | **-55.96 ± 8.79** | -4222.17 ± 1005.94 | -4209.67 ± 1832.02 | -1678.42 ± 902.38 | -1382.93 ± 465.56 |
| | 4 | **-61.89 ± 21.14** | -5697.90 ± 1325.49 | -9202.50 ± 1737.65 | -4080.83 ± 1036.82 | -2667.19 ± 1314.90 |
| | 6 | **-55.38 ± 13.66** | -12620.99 ± 1520.22 | -21224.53 ± 4607.58 | -9052.84 ± 3938.58 | -3453.83 ± 1829.18 |
| | 8 | **-168.15 ± 17.45** | -11231.50 ± 1670.98 | -20313.26 ± 2997.75 | -9260.05 ± 3438.89 | -5492.45 ± 3658.35 |
| Insurance | 2 | **-25.86 ± 0.44** | -1164.88 ± 516.83 | -40.20 ± 6.16 | -32.62 ± 5.92 | -38.57 ± 6.73 |
| | 4 | **-53.72 ± 1.22** | -2243.02 ± 1028.09 | -84.54 ± 8.04 | -71.41 ± 10.06 | -84.30 ± 8.75 |
| | 6 | **-74.35 ± 3.99** | -3043.59 ± 1869.63 | -221.84 ± 42.35 | -134.43 ± 14.37 | -135.66 ± 21.98 |
| | 8 | **-102.43 ± 11.05** | -4575.26 ± 1773.28 | -215.94 ± 28.86 | -151.45 ± 19.72 | -212.57 ± 45.98 |
| Hailfinder | 2 | **-72.26 ± 4.91** | -2114.56 ± 862.63 | -98.37 ± 2.29 | -95.73 ± 5.22 | -96.02 ± 2.19 |
| | 4 | **-159.04 ± 12.43** | -4087.08 ± 2267.90 | -212.83 ± 8.75 | -206.99 ± 4.21 | -213.92 ± 10.55 |
| | 6 | **-242.05 ± 18.18** | -6795.57 ± 2304.71 | -3297.89 ± 2712.97 | -318.81 ± 17.89 | -2027.16 ± 1159.21 |
| | 8 | **-301.92 ± 7.11** | -8087.45 ± 2835.30 | -4284.77 ± 2012.99 | -1744.57 ± 673.88 | -3516.97 ± 1476.38 |
| Hepar2 | 2 | **-69.21 ± 0.07** | -95.25 ± 8.65 | -82.35 ± 2.36 | -79.12 ± 3.08 | -74.64 ± 4.37 |
| | 4 | **-138.29 ± 0.00** | -173.10 ± 6.61 | -155.38 ± 4.51 | -156.43 ± 5.29 | -144.97 ± 5.60 |
| | 6 | **-278.00 ± 0.07** | -312.63 ± 14.36 | -314.49 ± 7.09 | -306.27 ± 5.21 | -289.60 ± 7.78 |
| | 8 | **-312.14 ± 0.00** | -385.30 ± 19.41 | -367.64 ± 6.15 | -361.39 ± 11.29 | -316.89 ± 4.38 |

has an advantage when used to perform inference on more complex networks.

We believe that this increased performance is due to the representation of each swarm being based on the Markov blankets and the corresponding competition 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. Also, 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.

While the OSI method appears to outperform the other methods in terms of the log likelihoods of solutions found, it requires many more fitness evaluations than the other approaches. Figure 3 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 approach is higher for networks with a large number of nodes. This is because the OSI approach creates 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.

## VIII. Conclusions

We have presented an algorithm based on overlapping swarm intelligence to approximate solutions to the $k$-MPE problem. In our approach, a swarm is assigned to each node in the Bayesian network, and that swarm learns the state assignments for its node's Markov blanket. We performed a series of experiments to compare our approach to the methods proposed in [5], [6], [15], and [18]. The results of these experiments indicate that, while our approach is more computationally expensive than competing methods, it significantly outperforms the competing approaches in terms of the log likelihoods of the solutions found when used to perform inference on complex Bayesian networks.

For future work we will compare our approach to other traditional $k$-MPE algorithm such as the exact inference method proposed in [12] and the approximate inference algorithm described in [14]. We also plan to design a method to allow the learning process to be more distributed by removing the need for a global fitness evaluation and compare its performance to that of the method presented here. This extension is similar to the one developed in [28]. Also, because a swarm is associated with each node, the number of swarms can be very large for complex networks. We plan to explore alternative representations and competition strategies to reduce this complexity. For example, we could assign swarms to only a subset of the nodes in the network. Each swarm could learn the state assignments for the Markov blanket of its assigned node for some number of iterations $t$. After $t$ iterations have elapsed the swarm could be assigned to a new node. We will also begin developing the theory of the general OSI framework. We will show convergence of our approach and draw on the approaches in [19] and [20] to analyze the relationship between OSI convergence rate and the structure of the sub-swarms.

## References

[1] S. Shimony, "Finding MAPs for belief networks is NP-hard," *Artificial Intelligence*, vol. 68, pp. 399–410, 1994.

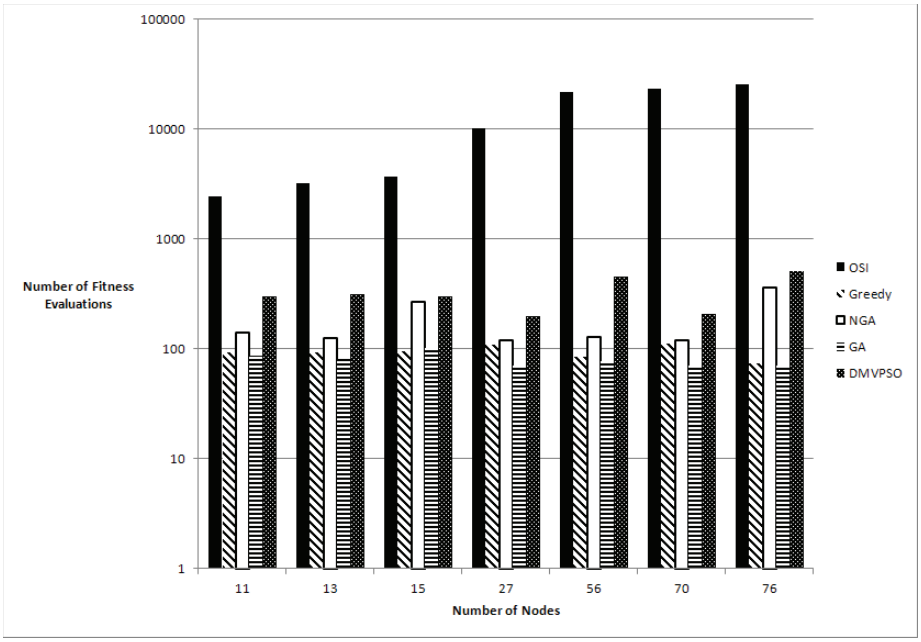



Fig. 3: Number of Fitness Evaluations


[2] P. Dagum and M. Luby, "Approximating probabilistic inference in Bayesian belief networks is NP-hard," *Artificial Intelligence*, vol. 60, no. 1, pp. 141–153, 1993.

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

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

[5] K. G. Pillai and J. W. Sheppard, "Abductive inference in Bayesian belief networks using swarm intelligence," in *Proceedings of SCIS-ISIS (to appear)*, 2012.

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

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

[8] G. F. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks," *Artificial Intelligence*, vol. 42, no. 2–3, pp. 393–405, 1990.

[9] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, pp. 33–57, 2007.

[10] J. Kennedy and R. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, vol. 5, oct 1997, pp. 4104–4108.

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

[12] D. Nilsson, "An efficient algorithm for finding the m most probable configurationsin probabilistic expert systems," *Statistics and Computing*, vol. 8, no. 2, pp. 159–173, 1998.

[13] A. Dawid, "Applications of a general propagation algorithm for probabilistic expert systems," *Statistics and Computing*, vol. 2, no. 1, pp. 25–36, 1992.

[14] E. Consortium *et al.*, "Elvira: An environment for creating and using probabilistic graphical models," in *Proceedings of the first European workshop on probabilistic graphical models*, 2002, pp. 222–230.

[15] E. Gelsema, "Abductive reasoning in Bayesian belief networks using a genetic algorithm," *Pattern Recognition Letters*, vol. 16, pp. 865–871, 1995.

[16] C. Rojas-Guzman and M. Kramer, "An evolutionary computing approach to probabilistic reasoning in Bayesian networks," *Evolutionary Computation*, vol. 4, pp. 57–85, 1996.

[17] L. de Campos, J. Gamez, and S.Moral, "Partial abductive inference in Bayesian belief networks using a genetic algorithm," *Pattern Recognition Letters*, vol. 20, pp. 1211–1217, 1999.

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

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

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

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

[22] R. Tanese, J. Co-Chairman-Holland, and Q. Co-Chairman-Stout, "Distributed genetic algorithms for function optimization," 1989.

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

[24] T. Belding, "The distributed genetic algorithm revisited," *arXiv preprint adap-org/9504007*, 1995.

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

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

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

[28] N. Fortier, J. W. Sheppard, and K. G. Pillai, "DOSI: Training artificial neural networks using overlapping swarm intelligence with local credit assignment," in *Proceedings of SCIS-ISIS (to appear)*, 2012.

[29] M. Scutari, "Bayesian network repository," 2012. [Online]. Available: http://www.bnlearn.com/bnrepository/