Abductive Inference in Bayesian Belief Networks Using Swarm Intelligence

Karthik Ganesan Pillai
Department of Computer Science
Montana State University
EPS 357, PO Box 173880
Bozeman MT, 59717-3880
Email: k.ganesanpillai@cs.montana.edu

John W. Sheppard
Department of Computer Science
Montana State University
EPS 357, PO Box 173880
Bozeman MT, 59717-3880
Email: john.sheppard@cs.montana.edu

***Abstract*—Abductive inference in Bayesian belief networks, also known as most probable explanation (MPE) or finding the maximum a posterior instantiation (MAP), is the task of finding the most likely joint assignment to all of the (non-evidence) variables in the network. In this paper, a novel swarm intelligence-based algorithm is introduced that efficiently finds the *k* MPEs of a Bayesian network. Our swarm-based algorithm is compared with two state-of-the-art genetic algorithms, and the results show that the swarm-based algorithm is effective and outperforms the two genetic algorithms in terms of computational resources required.**

***Index Terms*—Abductive inference, Bayesian networks, swarm intelligence**

I. Introduction

A Bayesian belief network is a directed acyclic graph (DAG), whose nodes are the random variables in a domain and whose edges correspond to direct influence of one node on another [11]. If $X = \{X_1, X_2 ... X_n\}$ is a set of variables in the network, then the joint probability can be calculated as

$$p(X) = \prod_{X_i \in X} p(X_i|\text{Pa}(X_i)), \tag{1}$$

where $\text{Pa}(X_i)$ corresponds to the parents of X_i. Bayesian belief networks provide a sound formalism for probabilistic reasoning under uncertainity [11]; however, one of the important challenges in using Bayesian networks is determining the values of unobserved variables in the domain that best explain the evidence. The task of abductive inference corresponds to computing the most likely joint assignment to all of the (non-evidence) variables in a network. More precisely, if we let $M = X \backslash E$, our task is to find the most likely assignment to the variables in M given the evidence $E = e$:

$$MAP(M, e) = \underset{m \in M}{\text{argmax}}\ p(m|e) \tag{2}$$

where, in general, $\text{argmax}_x f(x)$ represents the value of x for which $f(x)$ is maximal. There can be more than one assignment that has the highest posterior probability, and in this case the k-MPE task is to return the set of k most probable assignments. In [14], it has been shown that abductive inference in Bayesian belief networks is NP-hard. Abductive inference is a combinatorial optimization problem, and much research has gone into the possibilities of obtaining partial or approximate solutions.

Several authors have used genetic algorithms to approximate solutions for abductive inference [2], [4], [13], [15]. In [4], the state of a Bayesian network is represented by a chromosome where each chromosome is a string of integers. Each node in a network represents a bit position in a chromosome with possible values of 0 or 1. Standard genetic operators (i.e., mutation and crossover) are applied to these chromosomes to generate offspring from parent chromosomes. The chain rule is then applied to evaluate a chromosome's fitness since $p(M|e)$ is proportional to $p(M, e)$. Hence to evaluate a chromosome, $|M|$ multiplications are needed.

In [15], a niching genetic algorithm is introduced that utilizes the "multifractal characteristic and clustering property" of Bayesian networks to find k-MPE. The property corresponds to an observation that the joint probability distribution represented by a Bayesian network is skewed, and there are regions within the distribution that are highly "self-similar." This property led the authors to organize their GA with a novel probabilistic crowding method that biases the crossover operator toward self-similar individuals in the population. Otherwise, they followed the same approach as [4] to encode a Bayesian network into chromosomes.

In [13], the authors present a genetic algorithm that uses graphs for the chromosomes. Each chromosome encodes and completely specifies a possible solution that is equivalent to a complete description of a Bayesian network state. The phenotype in the solution space is the absolute probability of each set of assignments and constitutes an expression or interpretation of the encoding of the chromosome. Fitness is calculated as a transformation of the phenotype of each individual and is used to compare individuals.

Partial abductive inference is the task of finding k MPEs only for a subset of the network's variables, called an explanation set X_E [10]. In [2], a genetic algorithm was used to solve the partial abductive inference problem. Here, the explanation set variables are represented as a string of integers of length $|X_E|$. Each position in the string contains the state of the corresponding variable in X_E. Probabilistic propagation is then used to evaluate each chromosome, and calculations are carried out in the corresponding junction tree of the Bayesian network.

In this paper, we introduce a new approximation algorithm to solve the abductive inference problem based on particle swarm optimization, which is a swarm intelligence technique used to solve high-dimensional optimization problems. In particular, we introduce a discrete multi-valued PSO algorithm to find the k most probable explanations. We compare the results of our PSO-based approach to the genetic algorithms introduced in [4] and [15].

II. Background

A. Bayesian Belief Networks

Bayesian networks are also known as belief networks or causal diagrams. Bayesian networks consists of a set of propositional variables represented by nodes in a directed acyclic graph, and each variable can assume an arbitrary number of mutually exclusive and exhaustive values [13]. Directed arcs between nodes represent the probabilistic relationships between nodes. These directed arcs encode conditional independence properties in the Bayesian network. Specifically, we say a variables X_i is conditionally independent of its non-descendants given its parents.

$$\{X_i \perp \text{Non-Descendant}(X_i)|\text{Pa}(X_i)\}$$

Alternatively, we can say X_i is conditionally independent of all other variables in the network given its Markov blanket, where the Markov blanket corresponds to X_i's parents, children, and children's parents.

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

Prior probabilities are specified for each state of a root node, and conditional probabilities are specified for each possible value of a non-root node, given the the state of its parents [13].

Given a Bayesian network, several questions can be posed. For example, the standard inference problem involves providing a set of evidence E and then querying for the posterior probability given that evidence over some subset of variables $Q \subseteq X \backslash E$. As described previously, the k-MPE problem is an extension of the inference problem whereby we not only revise the posterior probability distribution based on the evidence but seek out the k most probable variable assignments that result. Given the base inference problem is NP-hard [1], its complexity is compounded by having to solve the resulting combinatorial optimization problem to find those k most probable state assignments [11], [14].

B. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a swarm intelligence technique, introduced by James Kennedy and Russel Eberhart, that is inspired by the social behavior of flocking birds [3]. PSO is a population-based approach where the algorithm is initialized with random solutions (called particles), and search applies an update process whereby the velocity vectors applied to the particles are determined based on the fitness of states visited by the particles. Eventually, all the particles in a swarm will move closer to an optimum of the fitness function. Originally, PSO was applied to optimization problems with continuous variables; however, several adaptations have been create to apply PSO to discrete problems, known as Discrete Particle Swarm Optimization (DPSO) [9]. In this paper, we use an extended version of DPSO, called discrete multi-valued particle swarm optimization [16], and apply the result to abductive inference problem in Bayesian networks.

1) Particle Swarm Optimization: The PSO algorithm first initializes a swarm of particles randomly over a search space. These particles "fly" with a certain velocity and find a position in the search space after each iteration. On every iteration of the algorithm, the current position of a particle is evaluated using the fitness function. The best position is stored in a vector called p_i (personal best). Also, the position of the particle with the best global fitness is stored in a vector called p_g (global best). At each iteration, the particle's velocity is updated based on the influence of the personal best position (p_i) and on the influence of the global best particle (p_g).

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

where $\otimes$ is component-wise multiplication and $\mathbf{U}(0, \phi)$ returns a vector of uniform random numbers between 0 and ϕ. Then each particle's position is updated by this newly calculated velocity:

$$x_i \leftarrow x_i + v_i.$$

There are a few parameters that need to be chosen for the PSO algorithm. The population size is chosen depending on the problem. The parameters ϕ_1 and ϕ_2 determine the force of the direction in which the particle is pulled between personal best and global best of the particles. These parameters need to be tuned properly for the PSO to converge. Also, the velocity of the particle is set to a minimum and maximum limit for the particle to control stability. To control the scope of the search, and also to control and perhaps eliminate the limit on velocity, an inertia weight ω is used.

2) Discrete Particle Swarm Optimization: For particles with binary-valued solution elements, the DPSO algorithm was developed by Kennedy and Eberhart [9]. In the DPSO algorithm, the position of each particle is represented by a vector from the d-dimensional binary solution space $x_i \in \{0, 1\}^d$, and velocity is a vector $v_i = (v_{i1}, v_{i2}, ...v_{id})$ of the d-dimensional continuous space, $v_i \in \Re^d$. Here the velocity term indicates the probability of the particle assuming a value of 0 or 1 in the next iteration. Specifically, the value v_{ij} equals the probability that x_{ij}, will take on the value of 1 in the next iteration. To assign a new position value to a particle i, each position variable x_{ij} is randomly set with probability of selecting a value of 1 given by the sigmoid function:

$$p(x_{ij} = 1) = \frac{1}{1 + \exp(-v_{ij})}$$

The DPSO algorithm is limited only to discrete problems with binary valued solution elements [9].

In [16], the discrete multi-valued PSO (DMVPSO) algorithm is introduced. In this method the discrete variables' values fall in the range $[0, M-1]$, where M corresponds to the cardinality of each state variable. The same velocity update and particle representation are used in this method as in DPSO. The position update equation for a particle is modified in the following manner. Using the following sigmoid transformation the velocity is transformed into a number between $[0, M]$:

$$S_{id} = \frac{M}{1 + \exp(-v_{id})}.$$

Given the fairly restricted range of values for v_{id}, the resulting curve is nearly linear. Thus, a new state value is determined by generating a random number according to the Gaussian distribution, $X_{id} \sim \mathcal{N}(S_{id}, \sigma \times (M-1))$, and rounding the result. Then to ensure the resulting value is legal, we truncate such that

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

This ensures the resulting positions of the particles remain discrete values between $[0, M-1]$. Notice that, for any given S_{id} there is a non-zero probability for choosing any value between $[0, M-1]$; however, because the Gaussian distribution is centered on S_{id}, the probability for selecting a given value decreases based on its distance from the current value of S_{id}.

III. Experimental Design

To solve the k-MPE problem using a PSO-based approach, we need to represent a candidate solution in the population as a particle in the corresponding swarm. In our case, a solution corresponds to a state assignment for all of the non-evidence variables within the Bayesian network. Thus we map this state vector to a particle by the following procedure.

The nodes in the network are assigned to individual state variables in an individual particle following a topological ordering of the network. The ordering is specified by beginning with root nodes and proceeding in a breadth-first fashion from left to right. This encoding is the same representation as the one used in [4]. We then need to define a fitness function so that we can evaluate each particle in the population during a given iteration. The objective of abductive inference is to find the most likely assignment to the variables in M given the evidence $E = e$ (see Equation 2). Therefore, we use the joint probability $p(m, e)$ as the fitness function with the goal of maximizing this probability. We justify this since $p(m|e) = p(m, e)/p(e)$ and $p(e) = 1$. This can be computing using Equation 1.

We compared our DMVPSO algorithm to two genetic algorithms in the literature that have been demonstrated to yield strong results on the abductive inference problem. The first algorithm is a basic adaptation of the standard genetic algorithm to the abductive inference problem and is described in [4]. Specifically, the chromosome in the genetic algorithm is a simple vector of state values for each of the variables in the network. For a Boolean network, this can be represented as a simple bit string; however, it is straightforward to generalize to any number of state values. The GA in [4] then uses fitness-proportionate selection with steady-state replacement. The genetic operators consist of simple mutation of individual states (except for evidence variables) and two-point crossover.

The GA in [4] was shown to perform well on the basic abductive inference problem; however, this was limited to the single best explanation. To handle the general k-MPE problem, we adapted their method by maintaining a priority queue of the k best explanations seen throughout the evolutionary process.

Another approach that has been developed to address the need to find multiple highly-fit individuals in a population is the niching genetic algorithm. Sriwachirawat and Auwatanamongkol propose and evaluate such an algorithm to solve the k-MPE problem [15]. Their "probabilistic restricted mating genetic algorithm" (PRMGA) adapts a probabilistic crowding procedure to promote crossover between similar individuals that also belong to the same cluster that happen to contain other high-fitness individuals (Algorithm 1). They also use a priority queue to keep track of the k best explanations, which provides a computational advantage over the simple GA.

Algorithm 1 Probabilistic Restricted Mating GA

```
for $i \leftarrow 1$ to $numGen$ do
  for $j \leftarrow 1$ to $popSize/2$ do
    $p_1 \leftarrow$ RandSelect(), $p_2 \leftarrow$ RandSelect()
    $pdist \leftarrow$ HammingDist($p_1$,$p_2$)$/\ell$
    if ($pdist \leq U(0,1)$) or ($Xover(p_1)|Xover(p_2)$) then
      $\{c_1, c_2\} \leftarrow$ Crossover($p_1$,$p_2$)
      $c_1 \leftarrow$ Mutate($p_1$), $c_2 \leftarrow$ Mutate($p_2$)
      $dist[1] \leftarrow$ HammingDist($c_1$,$p_1$)$/\ell$
      $dist[2] \leftarrow$ HammingDist($c_2$,$p_2$)$/\ell$
      $dist[3] \leftarrow$ HammingDist($c_1$,$p_2$)$/\ell$
      $dist[4] \leftarrow$ HammingDist($c_2$,$p_1$)$/\ell$
      if $\max(dist[1:4] \leq \theta)$ then
        $Xover(c_1) \leftarrow Xover(c_2) \leftarrow$ True
      else
        $Xover(c_1) \leftarrow Xover(c_2) \leftarrow$ False
      end if
      if $dist[1] + dist[2] \leq dist[3] + dist[4]$ then
        if ($U(0,1) \leq fit(c_1)/(fit(c_1) + fit(p_1))$) then
          Replace $p_1$ with $c_1$
        end if
        if ($U(0,1) \leq fit(c_2)/(fit(c_2) + fit(p_2))$) then
          Replace $p_2$ with $c_2$
        end if
      else
        if ($U(0,1) \leq fit(c_1)/(fit(c_1) + fit(p_2))$) then
          Replace $p_2$ with $c_1$
        end if
        if ($U(0,1) \leq fit(c_2)/(fit(c_2) + fit(p_1))$) then
          Replace $p_1$ with $c_2$
        end if
      end if
    else
      $c_1 \leftarrow$ Mutate($p_1$), $c_2 \leftarrow$ Mutate($p_2$)
      if ($U(0,1) \leq fit(c_1)/(fit(c_1) + fit(p_1))$) then
        Replace $p_1$ with $c_1$
      end if
      if ($U(0,1) \leq fit(c_2)/(fit(c_2) + fit(p_2))$) then
        Replace $p_2$ with $c_2$
      end if
    end if
  end for
end for
```

This algorithm works as follows. For all intents and purposes, the algorithm operates identically to the simple GA except for selection. Specifically, PRMGA uses the same type of mutation and can use any traditional crossover scheme (such as the two-point crossover used above). It also uses a generational replacement scheme (i.e., the entire population is replaced in each generation). For selection, however, two parents are simply selected randomly (without replacement) according to a uniform distribution. The similarity of the two parents (normalized by the length of the chromosome) is then determined and a random number is generated between 0 and 1. If either the distance between the parents is found to be

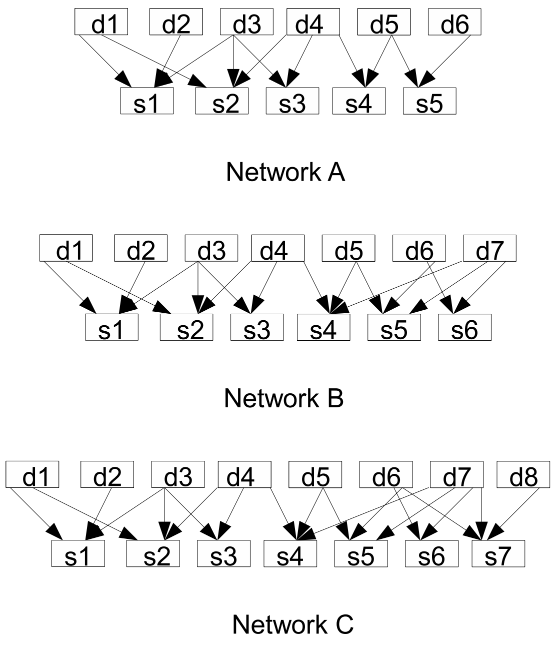

Figure 1. Bipartite Bayesian networks used for experiments

Table I
SEARCH SPACE PARAMETERS FOR FOR THE SIMPLE GA

states	evidence	nodes	card	npop	nsamp
2	2	A:11	512	8	24
		B:13	2048	32	96
		C:15	8192	128	384
	4	A:11	128	4	6
		B:13	512	8	24
		C:15	2048	32	96
3	2	A:11	19683	308	924
		B:13	177147	2768	8304
		C:15	1595323	24911	74733
	4	A:11	2187	34	102
		B:13	19683	308	924
		C:15	177147	2768	8304

less than this randomly generated threshold, or at least one of the two parents has their crossover flag set to true, then a crossover process begins. Otherwise, the parents simply undergo mutation.

Following mutation and crossover, the replacement process is based on the relative distances of parents and children as well as the relative fitness of parents and children. Specifically, the most similar parent-child pairs are determined, and their respective fitnesses are compared to determine whether the parent or the child survives. Limiting the replacement has the effect of limiting the reproduction/recombination process to those individuals and offspring that are most similar, thus creating the niches in the population.

To test the DMVPSO-based k-MPE algorithm, we used three different bipartite Bayesian networks (Figure 1) which we designate Network A, Network B, and Network C respectively. These networks are taken from [14]; however, the networks in [14] were specified using a noisy-OR model for the probabilities. Note that Cooper also proves the NP-hardness of inference, even with bipartite networks (including planar bipartite networks) [1]. Heckerman, on the other hand, provided an exact inference algorithm for bipartite noisy-OR networks (i.e., BN20 networks [8]) that is polynomial in the number of negative findings and exponential in the number of positive findings [6]. In our study, we decided to complicate the problem by assuming a traditional Bayesian network semantic rather than limiting the problem to the noisy-OR case.

For each of these networks, parameters for root nodes were generated randomly and parameters for non-root nodes were generated based on their parents and stored in tables. For each of these networks two different sets of parameters are generated. In the first parameter set, each node in the network has two states, and in the second parameters set, each node in the network has three states. As shown in Figure 1, Network A has six root nodes and five child nodes. Nodes s1 and s2 each have three parents while the remaining child nodes only have two parents. Network B has seven root nodes and six child nodes where s1, s2, s4, and s5 all have three parents. Finally Network C has eight root nodes and seven child nodes, and nodes s1, s2, s4, s5, and s7 all have three parents.

For each network and parameter set, twelve experiments were performed, based on the number of evidence nodes selected and number of k-MPEs. Separate experiments are run with two and four evidence variables set (where the evidence variables and their states are selected randomly). In addition, genetic algorithms, similar to one presented in [4] and [15], are used as a benchmarks for comparing the performance of DMVPSO [16]. For each network and set of evidence, we evaluated these three algorithms with k set to 2, 4, 6, and 8 respectively.

For both genetic algorithms, initial populations are generated by the following approach. Using probabilistic logic sampling [7], a configuration of all the variables in the network is generated by first setting the evidence variables X_e to their corresponding values. Note that any node in the network can serve as evidence using this method. For each non-evidence root node, that node's marginal probability is used to generate a state value, and for the remaining non-evidence nodes, their conditional probabilities are used to generate their state values based on the states of their parents. Thus a form of "forward sampling" is used to generate the state of each member of the population.

Table I shows the cardinality, population size, and total sample size for the two different parameter sets for the genetic algorithm of [4]. In this table, *card* is the cardinality of the search space (excluding evidence nodes), *npop* is the number of individuals in the population, and *nsamp* is the total number of individuals (including those in the initial population) explored during the entire genetic search process. The population size and number of samples are generated based on the cardinality of the search space. Specifically, $npop = round(card/64)$ and $nsamp = card/16 - npop$. The latter does not count the initial population. The parameters shown in this table were selected to be consistent with the approach taken in [4].

As in [4], we use two-point crossover and point mutation for the search operators. The population is updated using a steady-state replacement mechanism. In each generation two individuals are selected using fitness-proportionate selection. Of the two individuals created, one is selected randomly and subjected to mutation with a probability of 0.2. At the end of the reproduction cycle, as long as the resulting offspring does not match another member of the population, that offspring replaces the least fit individual in the population and a new reproduction cycle is started. The GA terminates when $nsamp$ individuals have been explored [4].

For the niching GA (NGA), the population was set to have 50 individuals. The mutation rate was set to 0.06, and the maximum number of generations was limited to 600. These parameters were used for both the 2-state and 3-state problems. The use of a static set of parameters for all problems is consistent with the approach taken in [15].

For DMVPSO, the evidence and initial population of particles are generated identically to the genetic algorithm populations. For all the experiments, the number of particles is set to 20, the maximum number of iterations is set to 100, and σ is set to 0.2.

The k-MPE solutions are generated similarly to the approach used in [15]. For each generation, the k best solutions are maintained in a priority queue. If a newly created individual does not exist in the queue and its fitness is higher than the lowest one in the queue, the lowest one is removed from the queue and the new individual is inserted into the queue instead. Each case (consisting of a network, a number of evidence nodes, and a value for k) is run 10 times, and the k most fit at the end of each of the 10 runs are returned as the recommended explanations.

Table II
PROBABILITIES OF THE k MOST PROBABLE EXPLANATIONS

Net	States	Evidence	KPE	DMVPSO	GA	NGA
A	2	2	2	0.0687	0.0687	0.0687
			4	0.1215	0.1215	0.1215
			6	0.1624	0.1624	0.2138
			8	0.2011	0.2011	0.2061
		4	2	0.1572	N/A	0.2201
			4	0.3394	N/A	0.3774
			6	0.3594	N/A	0.4975
			8	0.4876	N/A	0.4876
	3	2	2	0.0079	0.0080	0.0060
			4	0.0149	0.0147	0.0115
			6	0.0177	0.0177	0.0126
			8	0.0241	0.0255	0.0193
		4	2	0.0197	0.0197	0.0197
			4	0.0482	0.0482	0.0482
			6	0.0670	0.0670	0.0635
			8	0.0836	0.0836	0.0808
B	2	2	2	0.0401	0.0401	0.0401
			4	0.0665	0.0665	0.0065
			6	0.0950	0.0950	0.0950
			8	0.1116	0.1116	0.1078
		4	2	0.0715	0.0715	0.0572
			4	0.1318	0.1318	0.1318
			6	0.1696	0.1696	0.1696
			8	0.2047	0.2047	0.2047
	3	2	2	0.0011	0.0011	0.0006
			4	0.0019	0.0022	0.0013
			6	0.0024	0.0030	0.0022
			8	0.0032	0.0039	0.0022
		4	2	0.0042	0.0042	0.0038
			4	0.0077	0.0078	0.0068
			6	0.0113	0.0114	0.0083
			8	0.0145	0.0147	0.0128
C	2	2	2	0.0190	0.0190	0.0190
			4	0.0323	0.0323	0.0323
			6	0.0440	0.0440	0.0335
			8	0.0544	0.0544	0.0496
		4	2	0.0386	0.0386	0.0386
			4	0.0513	0.0513	0.0502
			6	0.0878	0.0898	0.0771
			8	0.1110	0.1110	0.0998
	3	2	2	0.0002	0.0002	0.0001
			4	0.0003	0.0005	0.0003
			6	0.0004	0.0007	0.0002
			8	0.0005	0.0007	0.0002
		4	2	0.0006	0.0007	0.0006
			4	0.0015	0.0015	0.0011
			6	0.0018	0.0020	0.0011
			8	0.0025	0.0029	0.0012

IV. Results and Discussion

The sums of joint probabilities of the $\{2, 4, 6, 8\}$-MPEs and time taken for two genetic algorithms and DMVPSO is shown in Tables II and III respectively. Note that these tables do not show results for Network A when the number of states for each variable is limited to 2 and there are 4 evidence nodes. This is because "nsamp" is extremely small relative to the number of explanations being sought. In particular, "nsamp" $=$ 6 while $k \in \{2, 4, 6, 8\}$. Thus in the best case, one third of the population will comprise the "best explanation." In one case, all of the members of the population must be used, and in one case, there are insufficient instances to cover the required number of explanations.

From Table III, we can observe as the network size and number of states increases, the time taken to run each of the genetic algorithms is significantly greater than the time required for DMVPSO. However, all three algorithms perform equally well on the sum of joint probabilities of the $\{2, 4, 6, 8\}$-MPEs.

V. Conclusions and Future Work

Our DMVPSO-based algorithm is an effective alternative for computing k-MPE efficiently. Specifically, we found the DMVPSO method is able to discover the k most probable explanations for our test networks that were statistically indistinguishable from the GA-based methods in the literature. Furthermore, our method was able to find these explanations in time that appears to scale linearly with the number of nodes in the network, independent of the number of explanations required. Furthermore, the corresponding times are dramatically superior to the GA whenever fewer evidence nodes have been set (corresponding to fewer constraints resulting from inference) and are superior to the NGA algorithm in all cases, even though the corresponding complexity appears to be linear in the number of explanations required. Effectiveness of the newly introduced algorithm is expected to increase on larger Bayesian networks.

For future work, we will compare the results of these algorithms, both in terms of accuracy and computational burden, with more traditional k-MPE algorithms. we will also consider larger networks as well as networks that are not restricted to being bipartite. Finally, recent work by Haberman and Sheppard [5] as well as Ganesan Pillai and Sheppard [12] in applying overlapping swarm-based methods will be adapted to determine if even better results in terms of accuracy and computational efficiency can be obtained.

References

[1] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

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

[3] R.C. Eberhart and J. Kennedy. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, 1995.

Table III
TIME IN MSEC FOR EACH ALGORITHM TO TERMINATE WITH THE k MOST PROBABLE EXPLANATIONS

Net	States	Evidence	KPE	DMVPSO	GA	NGA
A	2	2	2	74	12	2495
			4	65	7	2748
			6	65	4	2753
			8	63	6	2803
		4	2	54	N/A	3119
			4	56	N/A	3688
			6	55	N/A	3143
			8	56	N/A	3122
	3	2	2	73	1701	2230
			4	68	1781	2224
			6	67	1757	2208
			8	66	1630	2214
		4	2	58	32	2174
			4	58	31	2192
			6	55	30	2179
			8	57	27	2212
B	2	2	2	78	39	2402
			4	77	40	2792
			6	78	38	3389
			8	77	37	3674
		4	2	67	5	2352
			4	67	4	2508
			6	68	5	2799
			8	68	5	3663
	3	2	2	95	119608	2419
			4	82	129935	2994
			6	81	117968	2436
			8	82	118204	2409
		4	2	68	1420	2419
			4	67	1356	2372
			6	69	1287	2475
			8	70	1350	2378
C	2	2	2	89	556	2556
			4	89	489	2565
			6	90	499	2563
			8	89	502	2563
		4	2	80	33	2540
			4	80	36	2647
			6	80	33	2649
			8	81	34	2573
	3	2	2	94	12492893	2589
			4	95	11829822	2595
			6	96	11961393	2603
			8	96	11557441	2605
		4	2	86	130362	2560
			4	86	126711	2571
			6	86	122218	2554
			8	85	125177	3051

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

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

[6] D. Heckerman. A tractable inference algorithm for diagnosing multiple diseases. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence*, pages 174–181, 1989.

[7] M. Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In J. Lemmer and L. Kanal, editors, *Proceedings of the Second Conference on Uncertainty in Artificial Intelligence*, pages 149–163, 1986.

[8] M. Henrion and M. Druzdel. Qualitative propagation and scenario-based explanation of probabilistic reasoning. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 10–20, 1990.

[9] J. Kennedy and R. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, volume 5, pages 4104–4108, 1997.

[10] R. E. Neopolitan. *Probabilistic Reasoning in Expert Systems. Theory and Algorithms*. Wiley/Interscience, 1990.

[11] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc. San Francisco, CA, 1988.

[12] K. Ganesan Pillai and J. Sheppard. Overlapping swarm intelligence for training artificial neural networks. In *Proceedings of the IEEE Swarm Intelligence Symposium*, April 2011.

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

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

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

[16] 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*, pages 141–149, 2007.