Factored Evolutionary Algorithms

Shane Strasser, *Member, IEEE*, John Sheppard, *Fellow, IEEE*,
Nathan Fortier, *Member, IEEE*, and Rollie Goodman, *Member, IEEE*

***Abstract*—Factored evolutionary algorithms (FEAs) are a new class of evolutionary search-based optimization algorithms that have successfully been applied to various problems, such as training neural networks and performing abductive inference in graphical models. An FEA is unique in that it factors the objective function by creating overlapping subpopulations that optimize over a subset of variables of the function. In this paper, we give a formal definition of FEA algorithms and present empirical results related to their performance. One consideration in using an FEA is determining the appropriate factor architecture, which determines the set of variables each factor will optimize. For this reason, we present the results of experiments comparing the performance of different factor architectures on several standard applications for evolutionary algorithms. Additionally, we show that FEA's performance is not restricted by the underlying optimization algorithm by creating FEA versions of hill climbing, particle swarm optimization, genetic algorithm, and differential evolution and comparing their performance to their single-population and cooperative coevolutionary counterparts.**

***Index Terms*—Differential evolution (DE), genetic algorithm (GA), NK landscapes, particle swarm optimization (PSO).**

I. Introduction

MANY important problems require optimization, including bin packing, the traveling salesman problem, job shop scheduling, neural network training, and Bayesian network inference [1]. Often, stochastic search algorithms are used to solve such problems because the randomness used in the algorithms helps the algorithm to escape local optima. One of the best-known families of stochastic search algorithms is evolutionary algorithms (EAs).

In this paper, we present an extension of EA called factored evolutionary algorithms (FEAs) that has been found to perform very well in finding high quality solutions. FEA factors the optimization problem by creating overlapping subpopulations that optimize over subsets of variables. Before introducing the details of FEA, we first provide background information on some of the most commonly used EAs.

One of the best-known EAs is the genetic algorithm (GA) which is inspired by the idea of Darwinian evolution. Each individual in a GA acts like a chromosome and is modified in a manner that mimics genetics [2]. During each iteration, candidate solutions undergo operations such as reproduction and mutation. In reproduction, the candidate solutions first are selected for duplication and then reproduce using crossover. Mutation occurs after reproduction by changing randomly selected dimensions in the individual to some other value.

Differential evolution (DE) is another population-based algorithm that has been found to perform well on a variety of optimization problems [3]. The DE algorithm is similar to GA in that individuals undergo mutation, crossover, and selection. During mutation, a dimension of the individual's mutation vector is calculated using a weighted sum and difference of three unique randomly selected individuals. Next, crossover creates a trial vector for an individual by combining parts of the individual's mutation vector with its current position. Finally, in selection, the fitness of the new individual is compared to the current location. If the fitness is better, the individual's current position is set equal to the trial vector, but if the fitness is worse, the individual remains unchanged.

Another population-based approach to optimizing a function is called particle swarm optimization (PSO) [4]. Whereas GA and DE use a population of individuals that reproduce with one another, PSO uses a swarm of particles that "fly" around the search space. In addition to a vector that represents a candidate solution, particles use a velocity vector to control how the particles move in the search space. Each particle keeps track of its own best position found in a vector and the best position discovered by the entire swarm. During each update, a particle's position is moved toward its own best position and the best position in the swarm.

While GAs and DE have been applied successfully to a wide range of problems, they are susceptible to hitchhiking, which is when poor values become associated with good schemas [5], [6]. Similarly, PSO can be prone to what is called "two steps forward and one step back," which happens when near optimal parts of a individual's current position may be thrown away if the rest of an individual's position causes the individual to have low fitness [7]. Much of the existing work done to avoid hitchhiking and two steps forward and one step back in EAs is focused on maintaining diversity in the individuals [8]. For example, Parsopoulos and Vrahatis [9] presented an extension to PSO in which the swarm uses the concept of "repulsion" to keep individuals away from the current global and local best solutions. However, these approaches only address the symptoms of hitchhiking, such as convergence to suboptimal solutions. We propose an extension of EAs called FEAs that helps directly mitigate hitchhiking and two steps forward and one step back in GA, DE, and PSO.

Manuscript received September 22, 2015; revised January 14, 2016, April 20, 2016, and July 11, 2016; accepted August 15, 2016. Date of publication August 24, 2016; date of current version March 28, 2017. *(Corresponding author: Shane Strasser.)*

The authors are with the Department of Computer Science, Montana State University, Bozeman, MT 59717 USA (e-mail: shane.strasser@msu.montana.edu).

Digital Object Identifier 10.1109/TEVC.2016.2601922

The idea behind FEA is similar to how polynomials can be decomposed into a product of factors. FEA decomposes the optimization problem into a set of subpopulations, or factors that, when put together, represent full solutions to the problem. Additionally, FEA encourages the factors to overlap with one another, which allows the factors to compete with one another for inclusion in the full solution [10]. There are three main functions that FEA uses: 1) solving; 2) competition; and 3) sharing. The first is the solving or optimization step, which iterates over all of the factors and allows each factor to optimize over its variables. Next, FEA performs competition by finding sets of values from each factor that create a full solution with a high fitness. Finally, the algorithms performing sharing by using the full solution to inject information into the factors.

FEA is a generalized framework inspired by an earlier algorithm known as overlapping swarm intelligence (OSI), which is similar to PSO except that it uses overlapping subswarms. OSI has been used to train deep neural networks and to perform abductive inference in Bayesian networks, a type of probabilistic graphical model that is highly useful for reasoning under conditions of uncertainty in a multitude of applications [10], [11]. Here, by extending OSI into FEA, we allow any evolutionary or swarm-based algorithm to be utilized as the underlying optimization algorithm, broadening the range of applications in which a factored approach may be successful.

In this paper, we provide three primary contributions related to FEA. First, we give a detailed, formal definition of the FEA algorithm. Second, we conduct experiments exploring the relative merits of different factor architectures, i.e., methods of creating the subpopulations in the algorithm, in order to demonstrate the important of these architectures to the algorithm's performance. The proposed architectures are tested on several difficult problems from the literature: abductive inference in Bayesian networks, NK landscapes, which are a mathematical framework that generates tunable fitness landscapes for evaluating EAs, and a set of 7 benchmark optimization functions. In each case, the architectures tested are tailored to the type of problem to which FEA is being applied. Third, we conduct another set of experiments intended to demonstrate that the FEA approach can make use of a variety of underlying search algorithms. FEA variants using each of four underlying EAs—a GA, DE, PSO, and hill climbing (HC), either the continuous or discrete variants thereof depending on the application—are applied to the three problems listed above. For each variant, its performance is compared to the single-population and cooperative coevolutionary counterparts of the same underlying algorithm.

The remainder of this paper is organized as follows. We first discuss the related work in Section II and then give a detailed description of FEA in Section III. Section IV contains the necessary background and experiments regarding different factor architectures. Finally, we compare FEA to single population algorithms and cooperative coevolutionary (CC) algorithms in Section V and provide our conclusions and future work in Section VI.

II. Related Work

CC algorithms, some of the earliest algorithms that subdivide an optimization problem in an evolutionary setting, were originally proposed by Potter and De Jong [12]. In that work, Potter and De Jong [12] developed an algorithm called the cooperative coevolutionary GA (CCGA) that uses *subspecies* to represent nonoverlapping subcomponents of a potential solution. Complete solutions are then built by assembling the current best subcomponents of the subspecies. This paper showed that in most cases CCGA significantly outperformed traditional GAs. Only in cases where the optimization function had high interdependencies between the function variables (i.e., epistasis) did CCGA struggle because relationships between variables were ignored.

More dynamic versions of CCGA have been proposed that allow for subpopulations to evolve over time [13]. When stagnation is detected in the population, a new subpopulation is initialized randomly and added to the set of subpopulations. Similarly, a subpopulation is removed if it makes small contributions to the overall fitness. Because of the dynamic subpopulations, there does exist the possibility that two subpopulations may overlap with another. However, there is no guarantee that subpopulations will overlap, and the algorithm does not have a function to resolve discrepancies between overlapping subpopulations. Potter and De Jong [13] were able to demonstrate that their algorithm could evolve the correct number of subpopulations and was competitive with domain-specific algorithms on training cascade networks.

This idea of CC algorithms was also extended by Van den Bergh and Engelbrecht [7] to use PSO to train neural networks. In their paper, Van den Bergh and Engelbrecht [7] tested four fixed subpopulation architectures of their own design. Comparing these four different architectures, the success of the algorithms was highly dependent on the architecture used, due to the interdependencies between the variables. By keeping variables with interdependencies together, the algorithm was more effective at exploring the search space [7].

Later, Van den Bergh and Engelbrecht [14] extended their work by applying it to a wider range of optimization problems. Cooperative PSO (CPSO) was introduced as a generalization of the authors' prior work, which was able to get around the problem of losing good values since each dimension is optimized by a single subpopulation. However, one drawback to CPSO is that it can become trapped in what the authors call *pseudominima*, which are places that are minima when looking at a single dimension but over the entire search space are not local minima. To avoid this problem, Van den Bergh and Engelbrecht [14] described a hybrid algorithm that alternates between CPSO and PSO. The result was an algorithm that always outperformed PSO and was competitive with but more robust than CPSO.

CC algorithms have also been applied to DE. Shi *et al.* [6] proposed a simple extension of CCGA to DE, called CCDE. Other extensions have been more complex, such as those presented by Yang *et al.* [15], where Yang *et al.* [15] developed a weighted cooperative algorithm that used DE to optimize problems over 100 dimensions. This algorithm utilized a weighting scheme to allow for the evolution of subpopulations

where the function was optimized within the subpopulations. Yang *et al.*'s [15] algorithm was found to outperform regular CC algorithms on most of the test functions explored.

A variation of CC algorithms that used evolving subpopulations was also proposed by Li and Yao [16]. Here, the subpopulation sizes were allowed to grow or to shrink when stagnation was detected, creating a wider range of variable groups. Li and Yao [16] showed that their algorithm performed better than others on functions that had complex multimodal fitness landscapes, but performed slightly worse than PSO on unimodal functions. They noted that, while subpopulations of random variables perform well, there should exist more intelligent ways of creating subpopulations.

Similarly, this concept of optimizing over subsets of the problem has been applied to other domains, such as training neural networks. As a relatively new approach, Srivastava *et al.* [17] presented a variation of backpropagation called Dropout in which random parts of the neural network are deleted and the resulting "thinned" network trained. After training several thinned networks, these networks are combined together to create the full neural network. Srivastava *et al.* [17] found that Dropout outperformed standard backpropagation on full neural networks. In addition, Dropout performed well on other types of networks, such as restricted Boltzmann machines and deep belief networks.

OSI is a version of FEA that uses PSO as the underlying optimization algorithm. It was introduced in 2012 and works by creating multiple swarms that are assigned to overlapping subproblems [18]. It was first used as a method to develop energy-aware routing protocols for sensor networks that ensure reliable path selection while minimizing energy consumption during message transmission [18]. OSI was shown to be able to extend the life of the sensor networks and to perform significantly better than current energy-aware routing protocols.

The OSI algorithm was later extended by Pillai and Sheppard [11] to learn the weights of deep artificial neural networks. In that work, each swarm represents a unique path starting at an input node and ending at an output node. A common vector of weights is also maintained across all swarms to describe a global view of the network, which is created by combining the weights of the best particles in each of the swarms. Pillai and Sheppard [11] showed that OSI outperformed several other PSO-based algorithms as well as standard backpropagation on deep networks.

A distributed version of OSI was developed subsequently by Fortier *et al.* [19] called distributed OSI (DOSI). In that paper, a communication and sharing algorithm was defined so that swarms could share values while also competing with one another. The key distinction from OSI is that a global solution is not used for fitness evaluation. Fortier *et al.* [19] were able to show that DOSI's performance was close to that of OSI's on several different networks but there were several instances when OSI outperformed DOSI.

OSI and DOSI have also been used for inference tasks in Bayesian networks, such as abductive inference, where the task is to find the most probable set of states for some nodes in the network given a set of observations. Fortier *et al.* [10], [20] applied OSI and DOSI to perform full and partial abductive inference in Bayesian networks. Fortier *et al.* [10], [20] were able to show that OSI and DOSI outperformed several other population-based and traditional algorithms, such as PSO, GA, simulated annealing, stochastic local search, and mini-bucket elimination.

Other applications of OSI and DOSI include learning Bayesian networks. Fortier *et al.* [21] adapted OSI to learn the structure of Bayesian classifiers by allowing subswarms to learn the links for each variable in the network, where each variable represents an attribute in the data. For each variable in the network, two subswarms were created: one of the incoming links and other of the outgoing links. Fortier *et al.* [21] were able to show that in most cases OSI was able to significantly outperform the competing approaches.

When learning Bayesian networks, latent or unobserved variables are often introduced into the network. Fortier *et al.* [22] used OSI to learn the parameters of these latent variables. A subswarm was created for each node with unlearned parameters and all of the variables in that node's Markov blanket. Fortier *et al.* [22] were able to show that OSI outperformed the competing approaches and that the amount of overlap between the subswarms can impact the performance of OSI.

We also make note of the multifactorial EA (MFEA), introduced by Gupta *et al.* [23]. MFEA uses a single population to solve multiple optimization problems simultaneously. By optimizing different tasks that have additional influence over the search process, MFEA creates an exchange of information between the tasks. This creates an overlap between the tasks, which is similar in concept to how FEA uses sets of overlapping subpopulations. However, FEA's focus is on subdividing a single function (task) into overlapping sets of variables and using subpopulations to optimize over sets of variables. Because of the overlap in ideas, there does appear to be a relationship between FEA and MFEA. We leave exploration of FEA and MFEA as future work.

III. Factored Evolutionary Algorithms

FEAs are a new class of optimization algorithms that work by subdividing the optimization problem. FEA is similar to cooperative EAs like CCGA and CPSO; however, FEA encourages subpopulations to overlap with one another, allowing the subpopulations to compete and share information. FEA is also a generalization of the OSI algorithm since it allows for any EA to be used as the underlying optimization algorithm. This allows for FEA to be a general class of algorithms that includes CPSO, CCGA, OSI, and Island EAs. Here, we present a general definition of the FEA model that extends those presented in [10] and [11].

There are three major subfunctions in FEA: 1) solving; 2) competition; and 3) sharing. The solve function is the simplest and allows each factor to optimize over its set of variables. The competition function creates a full solution that is used by factors to evaluate a partial solution, while the sharing step uses the full solution to inject information in

the factors. Before giving the pseudocode for these steps, we first formally define factors in FEA.

A. Defining Subpopulations

Given a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ to be optimized with parameters $\mathbf{X} = \langle X_1, X_2, \ldots, X_n \rangle$, let $\mathbf{S}_i$ be a set of $\mathbf{X}$ of size k. Note that f can still be optimized over the variables in $\mathbf{S}_i$ by holding variables $\mathbf{R}_i = \mathbf{X} \backslash \mathbf{S}_i$ constant. A local subpopulation can then be defined over the variables in $\mathbf{S}_i$ that are optimizing f. An algorithm that uses a set of subpopulations to optimize a problem is called a multipopulation algorithm. We denote the set of s subpopulations in a multipopulation algorithm as $\boldsymbol{\mathcal{S}} = \cup_{i=1}^{s} \mathbf{S}_i$.

When $s = 1$ and $\mathbf{S}_1 = \mathbf{X}$, then $\boldsymbol{\mathcal{S}}$ will have just a single population that results in a traditional application of the population-based algorithm, such as PSO, DE, or GA. However, when $s > 1$, $\mathbf{S}_i \subset \mathbf{X}$, and $\bigcup \mathbf{S}_i = \mathbf{X}$ for all populations, the algorithm becomes a multipopulation algorithm. FEA is the case where there are subpopulations that are proper subsets of $\mathbf{X}$ and at least one subpopulation overlaps with another subpopulation.[1] Without loss of generality, assume every subpopulation overlaps some other subpopulation. Should there be disjoint subpopulations, we have a family of FEAs.

Because each population is only optimizing over a subset of values in $\mathbf{X}$, the subpopulation defined for $\mathbf{S}_i$ needs to know the values of $\mathbf{R}_i$ for local fitness evaluations. Given a subpopulation $\mathbf{S}_i$ and its remaining values $\mathbf{R}_i$, fitness for a partial solution in subpopulation $\mathbf{S}_i$ can be calculated as $f(\mathbf{S}_i \cup \mathbf{R}_i)$. The values for $\mathbf{R}_i$ are derived from the other subpopulations, which thereby allows $\mathbf{S}_i$ to use values optimized by other subpopulations. The algorithm accomplishes this through a competition step and sharing step as follows.

B. Competition

The goal of competition in FEA is to find the subpopulations with the state assignments that have the best fitness for each dimension. Here, we present the competition algorithm described in [10]. FEA constructs a global solution vector $\mathbf{G} = \langle X_1, X_2, \ldots, X_n \rangle$ that evaluates the optimized values from subpopulations. For every $X_i \in \mathbf{X}$, the algorithm iterates over every subpopulation containing X_i and finds the best value from those subpopulations. The competition algorithm is provided in Algorithm 1.

The algorithm first iterates over a random permutation of all the variables in $\mathbf{X}$, shown in line 2. Note that this permutation changes each time the algorithm is run. Lines 4 and 5 initialize variables that are used for the competition. Next, the algorithm iterates over another random permutation of all the subpopulations that are optimizing the variable X_i. Lines 10–15 then compare the individual values of variable X_i by substituting the subpopulations' values into $\mathbf{G}$. In our implementation, the factor uses the best value found during the entire search process as its candidate value to be evaluated in lines 10–15.

[1] If we do not require at least one overlap, other CC algorithms fit this definition.

Algorithm 1 FEA Compete Algorithm

Input: Function f to optimize, subpopulations $\boldsymbol{\mathcal{S}}$
Output: Full solution $\mathbf{G}$

```
1:  randVarPerm ← RandomPermutation(N)
2:  for ranVarIndex = 1 to n do
3:      i ← randVarPerm[ranVarIndex]
4:      bestFit ← f(G)
5:      bestVal ← S_0[X_i]
6:      S_i ← {S_k | X_i ∈ S_k }
7:      randPopPerm ← RandomPermutation(|S_i|)
8:      for ranPopIndex = 1 to |S_i| do
9:          S_j ← S_i[randPopPerm[ranPopIndex]]
10:         G[X_i] ← S_j[X_i]
11:         if f(G) is better than bestFit then
12:             bestVal ← S_j[X_i]
13:             bestFit ← f(G)
14:         end if
15:     end for
16:     G[X_i] ← bestVal
17: end for
18: return G
```

The values yielding the best fitness from the overlapping populations are saved and then inserted into $\mathbf{G}$. Once the algorithm has iterated over all variables in $\mathbf{X}$, the algorithm exits and returns $\mathbf{G}$.

Note that the competition algorithm is not guaranteed to find the best combination of values from each subpopulation, nor does it guarantee the combination of values is better than the previous $\mathbf{G}$. This is because, in most scenarios, finding the best combination of values will be as hard as the original problem. However, by iterating over random permutations of $\mathbf{X}$ and $\mathcal{S}$, the algorithm is able to explore different combinations and is still able to find good combinations of values. Additionally, a factor could use other sources to derive its candidate solution to be used during the competition phase, such as the current best individual in the factor's population. Finally, we note that because the if-statement in line 11 uses a strictly greater-than, the full global solution is only updated if the fitness improves. This eliminates scenarios where a variable oscillates between two different values with the same fitness, which also means the full global solution should converge to a single solution during the competition stage.

C. Sharing

The sharing step serves two purposes. The first is that it allows overlapping subpopulations to inject their current knowledge into one another. Previous work by Fortier *et al.* [10] discovered that this is one of the largest contributors to the FEA's performance. The second purpose of the sharing step is to set each subpopulation's $\mathbf{R}_i$ values to those in the full global solution $\mathbf{G}$ so that each subpopulation $\mathbf{S}_i$ can evaluate its partial solution on f. The sharing algorithm is provided in Algorithm 2.

The share algorithm iterates over all the subpopulations and sets each subpopulation's $\mathbf{R}_i$ values using the global

Algorithm 2 FEA Share Algorithm

Input: Full global solution $\mathbf{G}$, subpopulations $\boldsymbol{\mathcal{S}}$

1: **for all** $\mathbf{S}_i \in \boldsymbol{\mathcal{S}}$ **do**
2: $\quad \mathbf{R}_i \leftarrow \mathbf{G} \setminus \mathbf{S}_i$
3: $\quad \mathbf{p}_w \leftarrow \mathbf{S}_i$.worst()
4: $\quad \mathbf{p}_w \leftarrow \mathbf{G} \setminus \mathbf{R}_i$
5: $\quad \mathbf{p}_w.fitness \leftarrow f(\mathbf{p}_w \cup \mathbf{R}_i)$
6: **end for**
7: **return**

Algorithm 3 FEA

Input: Function f to optimize, optimization algorithm A
Output: Full solution $\boldsymbol{G}$

1: $\boldsymbol{\mathcal{S}} \leftarrow$ initializeSubpops($f, \mathbf{X}, A$)
2: $\mathbf{G} \leftarrow$ initializeFullGlobal($\boldsymbol{\mathcal{S}}$)
3: **repeat**
4: $\quad$ **for all** $\mathbf{S}_i \in \boldsymbol{\mathcal{S}}$ **do**
5: $\quad\quad$ **repeat**
6: $\quad\quad\quad \mathbf{S}_i$.updateIndividuals()
7: $\quad\quad$ **until** Termination criterion is met
8: $\quad$ **end for**
9: $\quad \mathbf{G} \leftarrow$ Compete($f, \boldsymbol{\mathcal{S}}$)
10: $\quad$ Share($\mathbf{G}, \boldsymbol{\mathcal{S}}$)
11: **until** Termination criterion is met
12: **return G**

solution $\mathbf{G}$ (line 2). These values are then used by $\mathbf{S}_i$ to evaluate its partial solutions. Next, the algorithm sets values in subpopulation $\mathbf{S}_i$ to those in $\mathbf{G}$. To accomplish this, the algorithm sets the current position of the individual with the worst fitness in $\mathbf{S}_i$'s population to the values in $\mathbf{G}$ (lines 3 and 4). The fitness is then recalculated in line 5 for the worst individual.

D. FEA Algorithm

Now, that the share and compete algorithms have been defined, we can give the full FEA algorithm, which is provided in Algorithm 3.

The algorithm works as follows. All of the subpopulations are first initialized according to the optimization algorithm being used and the subpopulation architecture (line 1). The full global solution $\mathbf{G}$ is initialized in line 2.

Next, the algorithm begins interpopulation optimization, which consists of three steps (lines 3–11). First, the algorithm iterates over each subpopulation and optimizes the values using the corresponding optimization algorithm until some stopping criterion is met (line 6). The optimization of each individual subpopulation is called the intrapopulation optimization step. Following intraoptimization of all subpopulations, competition occurs between subpopulations in the compete function on line 9. Finally, the share function on line 10 shares the updated best states between the subpopulations. The interpopulation optimization steps are repeated until the stopping criterion is met.

We note that while individual subpopulations may not converge to the same solution, previous work has found that in practice, the full global solution $\mathbf{G}$ converges to a single solution. This has yet to be proven formally; however, because the full global solution converges to a single solution during the competition step, FEA should also converge to a single solution. We leave the formal proof of convergence as future work.

IV. Factored Architecture Analysis

While there has been some work discussing different factor architectures, there has been little work to optimize these factor architectures for FEA. In this section, we verify empirically that the performance of FEA is tied to the factor architectures. To do so, we test a variety of architectures on three different problems: 1) abductive inference in Bayesian; 2) maximizing NK landscapes; and 3) optimizing of a set of commonly-used test functions. For each problem, we define a set of different factor architectures.

In our first set of experiments, we used PSO, as proposed by Kennedy and Eberhart [4], as the underlying algorithm within FEA for optimizing the benchmark test functions. For the discrete categorical optimization problems, NK landscapes and Bayesian networks, we used discrete multivalued PSO (DMVPSO), proposed in [24], as the underlying algorithm in order to compare our results with previously published results with OSI. We refer to this version of FEA as FEA-DMVPSO.

In DMVPSO, the velocity update equations remain mostly unchanged from regular PSO [4]. However, the semantics of the velocity vector are changed such that it denotes the probability of a particle's position taking on a specific value. After the velocity is updated, it is transformed into the interval $[0, M-1]$, where M is the number of values the variable may take on, using the sigmoid function $S_{i,j} = [M - 1/1 + \exp(-V_{i,j})]$. Next, each particle's position is updated by generating a random number according to the Gaussian distribution, $X_{i,j} \sim N(S_{i,j}, \sigma \times (M-1))$ and rounding the result. Then, a piecewise function is used to ensure all sampled values fall within the valid range $[0, M-1]$ [24]. In all our experiments, the PSO and DMVPSO ω parameters were set to 0.729, and ϕ_1 and ϕ_2 were both set to 1.49618. Each subpopulation for FEA had a population size of 10. We will now describe the three different sets of experiments and the results for each set.

A. Bayesian Networks

First, we define Bayesian networks [25]. Let $G = (\boldsymbol{X}, \boldsymbol{E})$ be a directed acyclic graph. $\boldsymbol{X}$ is the set of random variables in a joint probability distribution $P(X_1, \ldots, X_n)$ and $\boldsymbol{E}$ represents relationships between the random variables. Specifically, an edge $e_{i,j} \in \boldsymbol{E}$ means that X_i is conditionally dependent on X_j A joint distribution for a Bayesian network is then defined as

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

where $\mathrm{Pa}(X_i)$ corresponds to the parents of X_i. With this representation, a node's Markov blanket is given node's

TABLE I
PROPERTIES OF THE TEST BAYESIAN NETWORKS

Network	Nodes	Arcs	Parameters	Avg. MB Size
Hailfinder (Ha)	56	66	2656	3.54
Hepar2 (He)	70	123	1453	4.51
Insurance (I)	27	52	984	5.19
Win95pts (W)	76	112	574	5.92

parents, children, and its children's parents. A node is conditionally independent of all other nodes in the network given its Markov blanket.

Next, given a set of evidence variables and their corresponding states, abductive inference is the problem of finding the maximum *a posteriori* (MAP) probability state of the remaining variables of a network. If we let $\boldsymbol{X}_U = \boldsymbol{X} \backslash \boldsymbol{X}_O$, where $\boldsymbol{X}$ denotes the variable nodes in the network, the problem of abductive inference is to find the most probable state assignment to the unobserved variables in $\boldsymbol{X}_U$ given the evidence $\boldsymbol{X}_O = \boldsymbol{x}_o$

$$\text{MAP}(\boldsymbol{X}_U, \boldsymbol{x}_o) = \underset{\boldsymbol{x} \in \boldsymbol{X}_U}{\text{argmax}}\ P(\boldsymbol{x}|\boldsymbol{x}_O).$$

Note that multiplying small probabilities several times can cause the values to go to zero. To deal with this issue, the log is often used instead of the raw probabilities. Since taking the log does not change the optimization problem, the MAP equation becomes

$$\text{MAP}(\boldsymbol{X}_U, \boldsymbol{x}_O) = \underset{\boldsymbol{x} \in \boldsymbol{X}_U}{\text{argmax}} \sum_{i=1}^{n} \log P(X_i|Pa(X_i)).$$

1) Methodology: For all our experiments, we used an empty evidence set; therefore, we are searching for the most probable state assignment for all variables, i.e., $\mathbf{X}_U = \mathbf{X}$. While evidence could be applied to the network, setting evidence introduces another set of parameters into the experiments: which nodes in the network are assigned evidence. Additionally, when evidence is applied to the network, the number of variables to be optimized is reduced. Thus, by choosing not to apply evidence, we are testing on a more difficult optimization problem.

To test our architectures, we used the Hailfinder, Hepar2, Insurance, and Win95pts Bayesian networks from the Bayesian network repository [26]. These networks were chosen to be consistent with [10], for the sake of comparison. Table I lists the number of nodes, edges, parameters, and average Markov blanket size for the selected networks. For each Bayesian network, we compared four different factor architectures, which are described below.

a) Random: Random subpopulations are considered as the baseline architecture. For this approach, a random subpopulation is constructed for each of the N variables. M variables are then added to each of the N subpopulations. For each individual Bayesian network, M was set to be equal to the rounded average Markov blanket size for the network.

b) Parents: For each variable X_i, we construct a subpopulation of individuals consisting of $X_i \cup \text{Pa}(X_i)$. This is one of the simplest ways to subdivide a Bayesian network and provide overlap. Additionally, this architecture takes advantage of the structure of the log likelihood function that is used to evaluate the fitness of individuals in the population.

TABLE II
AVERAGE FITNESS OF DIFFERENT FACTOR ARCHITECTURES FOR FEA-DMVPSO PERFORMING ABDUCTIVE INFERENCE ON BAYESIAN NETWORKS

Network	Random	Parents	Markov	Clique
Hailfinder	-37.98±2.17	-36.81±2.08	-35.13±1.59	-95.88±204.36
Hepar2	-19.46±2.73	-17.16±1.38	-16.37±0.00	-16.57±0.83
Insurance	-12.96±2.24	-12.32±2.26	-11.61±1.81	-12.73±2.10
Win95pts	-39.02±5.60	-48.24±105.62	-33.25±5.95	-68.22±148.80

TABLE III
HYPOTHESIS TESTS COMPARING DIFFERENT FACTOR ARCHITECTURES FOR FEA-DMVPSO PERFORMING ABDUCTIVE INFERENCE ON BAYESIAN NETWORKS

Architecture	Random	Parents	Markov	Clique
Random	–	–	–	–
Parents	Ha,He,–,–	–	–	Ha,–,–,–
Markov	Ha,He,I,W	Ha,He,–,–	–	Ha,–,I,–
Clique	–,He,–,–	–,He,–,–	–	–

c) Markov: This architecture uses the Markov blanket of the nodes to create subpopulations, which offers arguably one of the most natural ways to subdivide a Bayesian network and provide overlap. Here, each subpopulation consists of $X_i \cup \text{Pa}(X_i) \cup \text{Ch}(X_i) \cup \text{Pa}(\text{Ch}(X_i))$ where $\text{Ch}(X_i)$ returns all children of X_i. In other words, the subpopulation consists of a node itself, and every node in its Markov blanket. This architecture may provide an advantage when performing inference because every node in the network is conditionally independent of all other nodes when conditioned on its Markov blanket.

d) Clique: The Clique Tree architecture is one of the more complicated methods to create subpopulations. The Bayesian network is first moralized, which consists of connecting parents of variables with an undirected edge. Next, the directed edges of the Bayesian network are made to be undirected, followed by triangulating the network. Finally, the graph is decomposed into a clique tree by first computing the maximal cliques and then finding the maximum spanning tree weighted by the overlap size of neighboring cliques. Note that unlike the previous architectures, this method does not build a subpopulation by centering on a single variable. Instead, each subpopulation corresponds to the variables in a clique in the resulting clique tree.

2) Results: Table II shows the results for the FEA-DMVPSO when performing abductive inference on the four different Bayesian networks using the four different factor architectures. The average fitness values, along with the standard deviation for error bounds, are displayed. Additionally, the results of statistical significance testing using a Paired Student t-test with $\alpha > 0.05$ between each pairs of factor architectures are shown in Table III. If an architecture performed significantly better than another architecture on a particular Bayesian network, the network's abbreviation is shown where the row architecture outperforms the column architecture. If there is no significant difference, a "–" is shown.

The Markov architecture outperformed all other architectures on all networks. However, it only significantly outperformed the Parents architecture on the Hailfinder and Hepar2 networks, and the Clique architecture on the Hailfinder and Insurance networks. The Parents architecture also performed well on all networks except the Win95pts network, whereas the Clique architecture performed well only on the Hepar2 and Insurance networks. Finally, the random architecture performed the worst on the Hepar2 and Insurance networks and did not significantly outperform any of the other architectures.

B. NK Landscapes

An NK landscape is a function $f : \mathcal{B}^N \rightarrow \mathbb{R}^+$ where $\mathcal{B}^N$ is a bit string of length N. K specifies the number of other bits in the string that a bit is dependent on. Given a landscape, the fitness value is calculated as $f(\mathbf{X}) = (1/N)\sum_{i=1}^{N} f_i(X_i, nb_K(X_i))$, where $nb_K(X_i)$ returns the K bits that are located within X_i's neighborhood. The individual functions are defined as $f_i : \mathcal{B}^K \rightarrow \mathbb{R}^+$, and are generally created randomly.

Note that there are multiple ways to define the neighborhood function. In this paper, we return the next K contiguous bits of the string starting at X_i. If the end of the string is reached, then the neighborhood wraps back around to the beginning of the string. Increasing K makes the variables more dependent on one another, resulting in more rugged landscapes [27].

1) Methodology: We tested each architecture on NK landscapes with parameters $N = 25, 40$, and $K = 2, 5, 10$. In our experiments, we compared the following factored architecture strategies on 50 randomly generated landscapes.

a) Random: Random subpopulations for NK landscapes are constructed similarly to the Random architecture used in abductive inference in Bayesian networks. A subpopulation is created for every variable, and M variables are then added to each of the N subpopulations. For these experiments we set M to be equal to K, giving each factor a size of $K + 1$.

b) Neighborhood: For each variable X_i, we create a subpopulation and add all the variables in the set $nb_K(X_i)$. This results in subpopulation sizes of $K + 1$.

c) Loci: The loci subpopulation extends the neighborhood architecture. Each variable X_i is still used to create a subpopulation along with the variables in $nb_K(X_i)$. Therefore, we add variable X_j to the subpopulation if $X_i \in nb_K(X_j)$. In other words, the subpopulation consists of all the variables in its neighborhood and all variables that contain X_i in their neighborhood. The neighborhood architecture will result in factors similar to those created by the Markov architecture for Bayesian networks. This creates subpopulations of size $2K+1$.

2) Results: Table IV shows the results the FEA-DMVPSO on maximizing NK landscapes using different factor architectures. Results from the hypothesis testing are in Table V, where a $+$ is shown if the row architecture is better than the column architecture. If there is no significant difference or the row architecture is worse than the column architecture, a $-$ is shown.

The neighborhood architecture always outperforms the Loci and Random architectures. However, it is only significantly better than the Loci architecture for $K = 5$ and $K = 10$. Random was almost always the worst and was significantly outperformed by Neighborhood and Loci, except when $N = 25, K = 10$, where random performed statistically better than the Loci architecture. A key observation from these results is the Neighborhood architecture is never significantly outperformed on any of the landscapes.

TABLE IV
AVERAGE FITNESS OF DIFFERENT FACTOR ARCHITECTURES FOR FEA-DMVPSO MAXIMIZING NK LANDSCAPES

N	K	Random	Neighborhood	Loci
25	2	18.01±0.78	18.41±0.76	18.42±0.75
	5	18.20±0.67	18.59±0.59	18.43±0.59
	10	17.81±0.58	18.03±0.58	17.67±0.57
40	2	28.51±1.25	29.34±1.18	29.37±1.13
	5	28.73±0.92	29.66±0.75	29.40±0.74
	10	28.04±0.82	28.81±0.75	28.25±0.72

TABLE V
HYPOTHESIS TESTS COMPARING DIFFERENT FACTOR ARCHITECTURES FOR FEA-DMVPSO MAXIMIZING NK LANDSCAPES

N	K	Architecture	Random	Neighborhood	Loci
25	2	Random	−	−	−
		Neighborhood	+	−	−
		Loci	+	−	−
	5	Random	−	−	−
		Neighborhood	+	−	+
		Loci	+	−	−
	10	Random	−	−	−
		Neighborhood	+	−	+
		Loci	+	−	−
40	2	Random	−	−	−
		Neighborhood	+	−	−
		Loci	+	−	−
	5	Random	−	−	−
		Neighborhood	+	−	+
		Loci	+	−	−
	10	Random	−	−	−
		Neighborhood	+	−	+
		Loci	+	−	−

C. Test Functions

In this section, we evaluate the performance of FEA on continuous optimization problems by applying the algorithm to a set of test functions commonly used in the literature for testing EAs and swarm algorithms.

1) Methodology: The seven functions selected for these experiments were the Ackley's, Dixon-Price, Exponential, Griewank, Rosenbrock, Schwefel 1.2, and Sphere functions. We used the same set of standard function ranges as presented in [28]. All of the problems are minimization problems with global minima of 0.0, except for the Exponential which has a minimum of -1.0. Additionally, all of the problems are scalable, meaning they can be optimized for versions of any dimension. In our experiments, we used functions of 50 dimensions. For each function, we compared the following factored architecture strategies.

a) Simple subswarm: Simple swarms are specified by two parameters, l and m, where l controls how many variables each subswarm optimizes over while m dictates how many variables each neighboring subswarm overlap. For example,

TABLE VI
AVERAGE FITNESS OF DIFFERENT FACTOR ARCHITECTURES FOR FEA-PSO MINIMIZING DIFFERENT BENCHMARK TEST FUNCTIONS

	CS-2	CS-5	CS-10	SS-4,2	SS-10,5
Ackley's	1.34E-14(1.03E-15)	1.30E-14(8.30E-16)	2.93E-2(2.93E-2)	5.86E-2(4.07E-2)	1.22E+1(1.00E+0)
Dixon-Price	7.99E-2(7.99E-2)	2.56E-1(7.99E-2)	3.63E+0(3.18E+0)	2.48E+0(4.24E-1)	1.33E+0(2.99E-1)
Exponential	-1.00E+0(6.18E-17)	-1.00E+0(6.18E-17)	-1.00E+0(6.18E-17)	-1.00E+0(6.18E-17)	-1.00E+0(6.18E-17)
Griewank	4.26E-3(2.04E-3)	4.00E-3(1.61E-3)	1.15E-2(3.85E-3)	2.15E-2(1.08E-2)	1.05E-1(2.73E-2)
Rosenbrock	4.57E+0(5.72E-1)	6.58E+0(2.75E+0)	3.98E+0(6.03E-1)	1.71E+1(5.21E+0)	6.29E+1(7.68E+0)
Schwefel	1.67E+4(1.88E+3)	8.45E+2(4.84E+2)	3.94E+2(3.32E+2)	1.97E+4(4.79E+3)	1.22E+4(3.05E+3)
Sphere	9.50E-42(8.87E-42)	9.74E-36(5.36E-36)	6.75E-32(3.37E-32)	9.69E-36(4.81E-36)	2.82E-19(2.62E-19)

values of 4 and 2 for l and m would create a subwarms $S_1 = \{X_1, X_2, X_3, X_4\}$, $S_2 = \{X_3, X_4, X_5, X_6\}$, $S_3 = \{X_5, X_6, X_7, X_8\}$, and so on. We will denote simple subswarm factor architecture with parameters l and m as SS-l, m.

b) Centered subswarm: The centered subswarm generates a subswarm for each variable in the optimization problem. For each subswarm, the next l variables are included in the subswarm. The algorithm starts with variable X_1 and adds the next l variables $\{X_2, X_3 \ldots X_{l+1}\}$. This is repeated for all variables. We note that in our implementation, we did not use any wrap-around upon reaching the end of the list of variables. Because of this, subswarms centered around variables toward the end of the list of variables will be smaller. We denote this architecture as CS-l.

2) Results: Table VI displays the results of the different factor architectures on minimizing the benchmark test functions. Results are displayed as average fitness over 30 trials with the standard error shown in parentheses. CS-2 outperformed all other architectures on the Dixon-Price and Sphere functions. It was outperformed by CS-5 on Ackley's and Griewank. CS-10 performed the best on Rosenbrock and Schwefel. All of the architectures performed equally well on the Exponential function. Neither of the SS architectures performed better than the CS architectures.

While the results of all pairwise significance tests are not shown, we note that in almost all cases, all of the CS architectures tied in terms of performance. Only on the Schwefel problem did CS-5 and CS-10 perform significantly better than CS-2. The difference between SS-4,2 and SS-10,5 was significant for all functions except on the Exponential, Schwefel, and Sphere. Similarly, the SC-2 and SC-5 were significantly different from SS-10,5 on all functions except the Exponential, Schwefel, and Sphere. Finally, we note that CS-2 was only significantly outperformed on the Schwefel function.

D. Analysis

From the Bayesian network results, the Markov architecture performed the best across all networks. However, it was only statistically better than the other architectures on certain networks. We believe this is because the size of FEA's factors also affect FEA's performance. This can be demonstrated by comparing Markov's performance on the Win95pts and Hailfinder networks. In the Win95pts network, the average Markov blanket size of the network is 5.92 while the average Markov blanket size in Hailfinder is 3.54. Markov failed to be statistically better than Clique and Parents on Win95pts, but was statistically better than all other architectures on the Hailfinder network. These results also demonstrate that the Markov architecture will usually perform the best on networks that have relatively small average Markov blankets (<6).

Within the NK landscape experiments, the best-performing architecture was Neighborhood. In almost all cases, Neighborhood outperformed Loci and Random. The Neighborhood architecture is similar to the Parents architecture for abductive inference in Bayesian networks. However, when $K = 2$, the Loci architecture, which is similar to the Markov architecture for Bayesian networks, performed better than the Neighborhood. This supports our previous claim that the performance of FEA's factors also depends on the size of the factors. When K became large, Loci's performance became worse as compared to Neighborhood's.

Based on the benchmark results, CS-2 was the best-performing architecture except on the Schwefel function. In that case, CS-5 and CS-10 performed the best. The Schwefel function output is dominated by the summation of factors, where the factor is composed of a product of input variables and the number of variables in the product ranges from 1 to n. This results in a function with variables that interact with one another to a large degree. Based on the results, we see that the factor architecture with larger subswarms, CS-10, is better able to capture the high level of variable interaction in the Schwefel function.

For the other benchmark functions, the problems are either completely separable, like the Sphere function, or are dominated by the summation of the product of only one or two variables, like Rosenbrock. In those cases, the factor architecture with small subswarms, CS-2, is capable of capturing the variable interaction. These results indicate that the best factor architectures are those that are appropriately sized for the product of variables in the benchmark function. Subswarms that are too large can lead to problems such as hitchhiking or "2 steps forward, 1 step backward," like regular PSO. However, if the subswarms are too small, then FEA loses its effectiveness because variable interactions may not be captured by the subswarms.

V. COMPARISON WITH OTHER ALGORITHMS

Based on the experiments in the previous section, we demonstrated that there are factor architectures that perform better than others under some conditions. However, those experiments only demonstrated FEA using PSO or DMVPSO as the underlying optimization algorithm. One of the advantages to FEA is its generality; any EA can be used within the factored framework. In this section, we compare four

TABLE VII
COMPARISON OF SINGLE POPULATION, CC, AND FEA ALGORITHMS ON ABDUCTIVE INFERENCE ON BAYESIAN NETWORKS

	Hailfinder	Hepar2	Insurance	Win95pts
DMVPSO	−5.05E+2(1.21E+2)	−5.29E+1(9.89E−1)	−2.54E+1(8.48E−1)	−1.80E+2(6.68E+1)
CC-DMVPSO	−1.16E+3(5.13E+2)	−2.00E+1(1.48E+0)	−1.76E+2(1.00E+2)	−4.99E+2(1.52E+2)
FEA-DMVPSO	**−3.45E+1(1.87E−1)**	**−1.75E+1(5.82E−1)**	**−1.07E+1(1.14E−1)**	**−1.43E+1(7.21E−1)**
DDE	−5.77E+1(1.76E+0)	**−1.98E+1(1.11E+0)**	−1.74E+1(1.04E+0)	−7.04E+1(3.11E+0)
CC-DDE	−4.13E+2(1.24E+2)	−2.35E+1(6.11E−1)	−1.74E+2(1.52E+2)	−1.86E+2(9.18E+1)
FEA-DDE	**−3.60E+1(5.68E−1)**	**−2.04E+1(9.79E−1)**	**−9.21E+0(9.11E−1)**	**−2.24E+1(1.70E+0)**
GA	−3.83E+1(1.58E+0)	−1.75E+1(4.35E−1)	−1.26E+1(8.73E−1)	**−3.11E+1(7.77E−1)**
CC-GA	−3.78E+1(5.67E−1)	−2.10E+1(1.07E+0)	−1.49E+1(2.28E+0)	−3.44E+2(2.10E+2)
FEA-GA	−3.67E+1(3.62E−1)	−1.98E+1(9.68E−1)	−1.19E+1(5.31E−1)	**−2.93E+1(1.57E+0)**
HC	−3.86E+1(6.91E−1)	**−1.67E+1(2.44E−1)**	**−1.14E+1(4.64E−1)**	−2.56E+1(1.09E+0)
CC-HC	−8.66E+2(2.37E+2)	−2.08E+1(1.30E+0)	−9.45E+1(7.76E+1)	−4.92E+2(2.33E+2)
FEA-HC	**−3.62E+1(5.66E−1)**	−1.81E+1(5.16E−1)	**−1.11E+1(8.00E−1)**	**−1.74E+1(9.59E−1)**

different versions of FEA that each use a different underlying optimization algorithm. Each algorithm is then compared with single-population and CC versions of the underlying algorithm.

A. Experiments

To demonstrate the general performance of FEA, we applied FEA to abductive inference in Bayesian networks, maximizing NK landscapes, and minimizing the benchmark test functions from Section IV using HC, GA, DE, and PSO as the underlying algorithms. On Bayesian networks and NK landscapes, we used discrete DE (DDE) and DMVPSO. For the Bayesian networks, we used the set of networks in Table I. NK landscapes were randomly generated using combinations of $N = 2, 5, 10$ and $K = 2, 5, 10$. For the benchmark optimization functions, we used the Ackley's, Dixon-Price, Exponential, Griewank, Rosenbrock, Schwefel 1.2, and Sphere functions.

For each Bayesian network, 50 trials were performed for each algorithm. Because NK landscapes are randomly generated, we generated 50 landscapes, and each algorithm was then run 50 times on each landscape. Similarly, 50 trials were performed on each benchmark function.

We used the DDE algorithm proposed in [29], which rounds the individual's position during fitness evaluations to handle integer values. An indexing scheme is then used to map the integer value to the discrete value [29]. During tuning, we found that a value of 0.25 for both DE and DDE's mutation rate and a differential factor of 0.55 performed the best. For the GA, tournament section and one-point crossover were used along with uniform mutation with a mutation rate of 0.02. PSO and DMVPSO used the same parameter values as the previous experiments in Section IV.

The FEA versions of the algorithms used the Markov (Loci) architecture for the Bayesian networks and NK landscapes. While the Neighborhood/Factor architectures outperformed Markov/Loci when there was a large amount of interaction between variables, we wanted to use the same architecture over all problems for consistency, and the Markov/Loci architectures still performed well over all problems. Additionally, the use of this architecture allows a more direct comparison to previously published results by [10]. For the benchmark functions, we used SC-2 on all functions except on the Schwefel function, which used CS-10. This was because SC-2 performed the best or tied on those functions while CS-10 performed the best on the Schwefel function. Each subswarm performed ten iterations before competition and sharing were performed.

For the CC versions of DMVPSO, GA, DDE, and HC, $N/2$ subpopulations optimizing over two variables each were created, since these parameters gave the best results during preliminary testing across all algorithms. These algorithms are similar to the CPSO-S_K model presented by Van den Bergh and Engelbrecht [7]. When evaluating the fitness of a subpopulation for the CC algorithm, we use the best known value from the other subpopulations to fill in the remaining values. While this is different than other implementations, such as those presented in [12], it uses the same source for values that FEA uses when it constructs the global solution $\mathbf{G}$.

FEA, CC and the single-population algorithms were given a total of 350 individuals, except for the HC versions, where only 75 individuals were used. These values were found to perform well during tuning for all algorithms. On the CC and FEA algorithms, individuals were distributed evenly across each of the subswarms. All algorithms were stopped once the best fitness did not improve after 15 iterations.

B. Results

Table VII shows the results of comparing single-population and CC algorithms to the FEA versions on performing abductive inference on the four different Bayesian networks. Bold values indicate a statistically significant difference between the single population, CC, and FEA versions of the corresponding underlying algorithms, using Paired Student t-tests with a 0.05 significance level. If two algorithms tied, both values are bolded. In all cases, the FEA versions of the algorithms performed better than the CC versions. However, FEA did not always perform better than the single population algorithms. For example, the single-population GA tied with the FEA version on all four networks. The single-population DDE tied with FEA-DDE on the Hepar2. FEA-HC was significantly outperformed by HC on Hepar2, but tied on the Hailfinder and Insurance networks. The only instance in which FEA was significantly outperformed by the single-population algorithm was HC on the Hepar2 network. Finally, FEA-DMVPSO significantly outperformed DMVPSO on all four networks.

TABLE VIII
COMPARISON OF SINGLE POPULATION, CC, AND FEA ALGORITHMS ON MAXIMIZING NK LANDSCAPES

	25			40		
	2	5	10	2	5	10
DMVPSO	1.76E+1(4.35E−2)	1.79E+1(3.59E−2)	1.80E+1(3.23E−2)	2.68E+1(8.03E−2)	2.68E+1(5.16E−2)	2.68E+1(4.01E−2)
CC-DMVPSO	1.84E+1(6.53E−2)	1.78E+1(8.72E−2)	1.72E+1(7.98E−2)	2.95E+1(1.16E−1)	2.77E+1(1.86E−1)	2.63E+1(1.34E−1)
FEA-DMVPSO	**1.87E+1(3.93E−2)**	**1.93E+1(4.28E−2)**	**1.92E+1(4.77E−2)**	**3.02E+1(7.82E−2)**	**3.06E+1(6.15E−2)**	**3.07E+1(4.41E−2)**
DDE	**1.88E+1(3.94E−2)**	1.88E+1(3.60E−2)	1.84E+1(3.84E−2)	2.98E+1(8.04E−2)	2.82E+1(5.32E−2)	2.76E+1(4.59E−2)
CC-DDE	1.83E+1(5.49E−2)	1.77E+1(9.00E−2)	1.70E+1(7.98E−2)	2.95E+1(8.36E−2)	2.79E+1(1.66E−1)	2.65E+1(1.49E−1)
FEA-DDE	1.86E+1(4.16E−2)	**1.92E+1(4.18E−2)**	**1.91E+1(4.20E−2)**	**3.00E+1(7.88E−2)**	**3.02E+1(6.23E−2)**	**3.02E+1(4.75E−2)**
GA	**1.88E+1(3.98E−2)**	**1.91E+1(3.54E−2)**	1.84E+1(5.74E−2)	**3.03E+1(7.06E−2)**	**3.00E+1(7.66E−2)**	2.73E+1(5.30E−2)
CC-GA	1.82E+1(5.25E−2)	1.78E+1(9.79E−2)	1.70E+1(7.89E−2)	2.94E+1(8.13E−2)	2.78E+1(1.62E−1)	2.67E+1(1.37E−1)
FEA-GA	1.85E+1(4.97E−2)	1.89E+1(4.28E−2)	**1.87E+1(4.09E−2)**	2.98E+1(7.82E−2)	**2.99E+1(6.77E−2)**	**2.96E+1(5.50E−2)**
HC	**1.88E+1(3.98E−2)**	**1.94E+1(3.25E−2)**	**1.90E+1(3.35E−2)**	**3.03E+1(7.05E−2)**	**3.01E+1(4.56E−2)**	2.94E+1(4.41E−2)
CC-HC	1.83E+1(6.85E−2)	1.74E+1(1.12E−1)	1.70E+1(8.17E−2)	2.92E+1(1.10E−1)	2.73E+1(1.81E−1)	2.63E+1(1.65E−1)
FEA-HC	1.86E+1(4.27E−2)	1.91E+1(4.50E−2)	**1.89E+1(4.43E−2)**	**3.00E+1(7.68E−2)**	**3.02E+1(5.84E−2)**	**3.00E+1(5.63E−2)**

TABLE IX
COMPARISON OF SINGLE POPULATION, CC, AND FEA ALGORITHMS ON OPTIMIZING BENCHMARK FUNCTIONS

	Ackley's	Dixon-Price	Exponential	Griewank	Rosenbrock
PSO	4.30E−14(3.22E−15)	6.76E−1(5.13E−3)	-9.97E−1(2.75E−4)	**1.01E−2(1.55E−3)**	6.09E+1(4.75E+0)
CC-PSO	5.27E−14(2.19E−15)	**1.07E−8(4.92E−10)**	-1.00E+0(6.99E−17)	**1.30E−2(7.08E−3)**	2.43E+1(1.40E+1)
FEA-PSO	1.37E−14(6.58E−16)	5.44E+0(3.75E+0)	**-1.00E+0(6.18E−17)**	9.10E−2(1.60E−2)	**2.99E+0(5.75E−1)**
DE	1.09E+1(9.90E−1)	2.42E+5(5.94E+4)	-8.50E−1(3.95E−2)	1.17E+0(1.71E−1)	4.18E+5(9.13E+4)
CC-DE	**1.53E−2(1.53E−2)**	1.84E+2(8.49E+1)	**-1.00E+0(6.18E−17)**	**9.51E−2(4.45E−2)**	3.20E+2(8.59E+1)
FEA-DE	1.49E−1(5.13E−2)	**3.86E+1(9.20E+0)**	**-1.00E+0(1.96E−8)**	**5.52E−2(1.22E−2)**	**1.31E+2(1.78E+1)**
GA	1.39E−1(1.11E−2)	1.47E+1(1.27E+0)	-9.66E−1(1.52E−3)	**2.06E−2(2.02E−3)**	2.06E+2(1.25E+1)
CC-GA	**2.21E−2(1.56E−3)**	**3.61E+0(5.52E−1)**	**-9.94E−1(5.06E−4)**	5.44E−2(7.90E−3)	**4.74E+1(1.58E+1)**
FEA-GA	**2.08E−2(1.21E−3)**	**4.43E+0(5.83E−1)**	**-9.94E−1(5.75E−4)**	9.88E−2(1.24E−2)	7.38E+1(1.10E+1)
HC	1.92E+1(4.10E−2)	1.89E+2(2.21E+1)	-6.11E−6(1.36E−6)	1.08E+0(3.40E−2)	8.23E+2(6.67E+1)
CC-HC	1.69E+0(7.93E−2)	1.22E+1(1.20E+0)	-9.99E−1(1.62E−4)	4.28E−1(8.34E−2)	6.53E+1(8.78E+0)
FEA-HC	**5.95E−3(4.60E−4)**	**1.15E+0(4.33E−1)**	**-1.00E+0(7.87E−7)**	**3.34E−2(7.94E−3)**	**2.07E+0(3.67E−1)**

The results comparing single-population, CC, and FEA algorithms on the NK landscapes are shown in Table VIII. Similar to the Bayesian network results, the FEA versions of the algorithms significantly outperformed the CC versions. FEA-DMVPSO always outperformed the single-population versions. However, FEA-DDE was outperformed on the simplest landscapes ($N = 25, K = 2$). GA significantly outperformed FEA-GA on $N = 25, K = 2$, $N = 25, K = 5$, and $N = 40, K = 2$, but tied on $N = 40, K = 5$. FEA-GA outperformed the single-population algorithm when $K = 10$. Looking at the HC algorithms, FEA outperformed the single-population algorithm only on $N = 40, K = 10$. The single population HC significantly outperformed the FEA version on $N = 25, K = 2$ and $N = 25, K = 5$. On the other landscapes ($N = 25, K = 10$, $N = 40, K = 2$, $N = 40, K = 5$), the FEA and single population HC algorithms tied.

Tables IX and X show the results of comparing single-population, CC algorithms, and FEA versions on optimizing the benchmark functions. Results are presented as the mean value over 30 trials along with the standard error shown in parentheses. Overall, the FEA versions of the algorithms performed the best. FEA-PSO was outperformed by CC-FEA on the Dixon-Price and Griewank functions. On the Ackley's function, CC-DE performed significantly better than FEA-DE. The GA outperformed FEA-GA on the Griewank function while CC-GA performed significantly better on the Rosenbrock function. FEA-HC performed significantly better than single-population and CC versions on all functions. Finally, there are several instances where the CC and FEA algorithms tied.

TABLE X
COMPARISON OF SINGLE POPULATION, CC, AND FEA ALGORITHMS ON OPTIMIZING BENCHMARK FUNCTIONS

	Schwefel	Sphere
PSO	**1.99E+3(3.32E+2)**	3.60E−3(3.11E−4)
CC-PSO	8.43E+5(2.03E+5)	**2.37E−17(6.24E−18)**
FEA-PSO	**1.81E+3(6.54E+2)**	**5.79E−17(5.23E−17)**
DE	1.15E+5(3.24E+3)	1.49E+3(4.49E+2)
CC-DE	9.45E+5(1.89E+5)	**2.11E−30(4.94E−31)**
FEA-DE	**9.18E+2(1.62E+2)**	**7.80E−3(7.80E−3)**
GA	9.26E+4(5.68E+3)	7.15E−2(8.09E−3)
CC-GA	1.90E+6(4.65E+5)	**9.81E−3(6.63E−4)**
FEA-GA	**1.07E+2(1.23E+1)**	**9.39E−3(9.23E−4)**
HC	3.66E+4(1.86E+3)	2.04E+2(1.33E+1)
CC-HC	1.54E+6(5.63E+5)	2.11E+0(2.32E−1)
FEA-HC	**9.71E+2(3.41E+2)**	**5.01E−3(3.88E−4)**

We also looked at the number of fitness evaluations each algorithm required. For the sake of conciseness, the combined results of the discrete and continuous problems are presented in Table XI. The "update" column gives the number of fitness evaluations each of the algorithms used while updating the individuals, while the "total" column presents the total number of evaluations used by the algorithm. We present these two different values because FEA requires extra fitness evaluations during the competition phase of the algorithm, and thus it may not be appropriate to consider only the total number of fitness evaluations. For example, the FEA-PSO uses 5.02E+05 fitness evaluations while updating the subpopulations. But during the competition phase, it performed an additional 3.80E+00 fitness evaluations, giving it a total of 5.41E+05 fitness evaluations.

TABLE XI
RESULTS COMPARING THE NUMBER OF FITNESS EVALUATIONS ON DISCRETE AND CONTINUOUS FUNCTIONS

	Discrete Functions		Continous Functions	
	Update	Total	Update	Total
PSO	2.69E+5	2.69E+5	1.11E+4	1.11E+4
CC-PSO	2.47E+5	2.47E+5	8.02E+4	8.02E+4
FEA-PSO	5.02E+5	5.41E+5	9.21E+4	1.02E+5
DE	3.05E+4	3.05E+4	1.83E+4	1.83E+4
CC-DE	1.51E+5	1.51E+5	9.12E+4	9.12E+4
FEA-DE	9.10E+5	1.02E+6	7.33E+4	8.12E+4
GA	8.14E+4	8.14E+4	2.64E+4	2.64E+4
CC-GA	8.48E+5	8.48E+5	9.16E+4	9.16E+4
FEA-GA	2.57E+6	2.85E+6	8.21E+4	9.17E+4
HC	1.10E+6	1.10E+6	9.90E+4	9.90E+4
CC-HC	6.40E+4	6.40E+4	3.86E+4	3.86E+4
FEA-HC	2.10E+6	2.23E+6	8.04E+4	8.81E+4

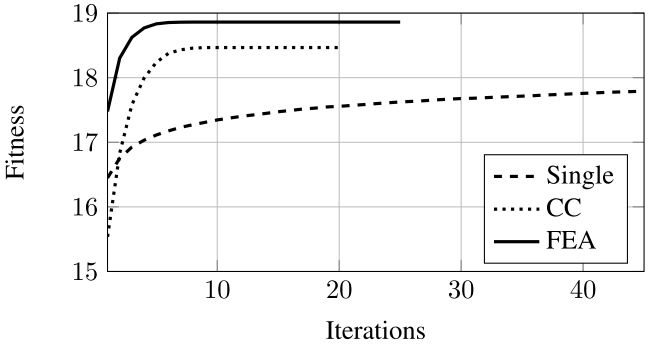

Fig. 1. Fitness curve plotting fitness versus iterations on single population, CC, and FEA DMVPSO algorithms maximizing NK landscapes with $N = 25$ and $K = 2$.

In almost all cases, the FEA algorithms required more fitness evaluations than the single-population and CC versions. On the continuous functions, CC-GA and CC-DE both performed more fitness evaluations than the FEA versions. HC also performed more fitness evaluations than FEA-HC on the continuous problems. While FEA does require additional fitness evaluations during the competition step, this number is small compared to the total number of fitness evaluations performed by FEA. We note that the extra number of fitness evaluations required during competition make up around 10% of the total number of fitness evaluations.

Finally, we present fitness curves from DMVPSO maximizing NK landscapes $N = 25, K = 2$ and PSO minimizing the Rosenbrock function. Fig. 1 presents the average best fitness over time on the NK landscape problem while Fig. 2 shows the same for the Rosenbrock function. The curve labeled "single" refers to the single population algorithm while "CC" and "FEA" refer to the CC and FEA versions, respectively. Note that the *y*-axis is on a log scale to allow for a compact representation of the results. Additionally, a single iteration for CC and FEA involves iterating over all subpopulations and allowing each subpopulation to run for ten iterations while an iteration for a single population is given after updating each of the individuals one time. All algorithms were stopped if, after 15 iterations, the best fitness failed to improve. For both figures, FEA and CC were both able to converge quickly to good values. The single-population algorithm took longer to converge, and also converged to a less-fit solution.

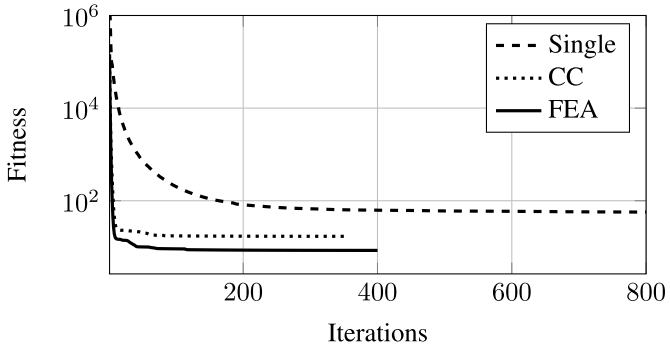

Fig. 2. Fitness curve plotting fitness versus iterations on single population, CC, and FEA PSO algorithms minimizing the Rosenbrock function. Note that the *y*-axis is shown on a logarithmic scale to allow for a compact display of fitness values over time.

C. Analysis

The Paired *t*-tests on the sum of the log likelihoods show that FEA versions of DMVPSO all performed significantly better than the corresponding single-population and CC versions. This demonstrates that FEA's use of overlapping subpopulations is more effective than the nonoverlapping subpopulations used by the CC algorithm.

Additionally, the results suggest that on difficult problems, such as the Hailfinder Bayesian network, FEA offers an increase in performance for almost all algorithms. We believe that this is because FEA's overlapping subpopulations help the algorithm avoid hitchhiking and two steps forward, one step back. If a subpopulation contains a poor value due to hitchhiking, a better value from a different subpopulation may be selected during the competition. This new value will then be injected into the other subpopulation during the share step of FEA, thus eliminating the value that is hitchhiking.

The results from the NK landscapes further support this hypothesis. FEA's performance over single-population and CC algorithms is significant on the landscapes with high variable interaction ($K = 10$). On those problems, the landscape is more complex, which increases the probability of the algorithms becoming trapped in local optimum. We believe that FEA is able to maintain more diversity during the search process because the subpopulations are updated independently during the solve step. This is a similar concept to the Island model for EAs, where (full) subpopulations are updated independently of one another.

Finally, the benchmark function results further support our hypothesis that FEA versions of population-based algorithms perform better than both single-population and CC algorithms on complex or difficult problems. Many of the test functions used are designed to be difficult problems with many traps and valleys. FEA allows the algorithm to escape these by breaking up variables that are hitchhiking and allowing the algorithm to maintain diversity between the subpopulations. This is especially noticeable when comparing HC and FEA-HC, where FEA-HC always outperformed HC. Because HC is a greedy algorithm that only moves in directions with better fitness, it is susceptible to becoming stuck in local optima. FEA-HC allows the individuals to escape those suboptimal locations by sharing information between those individuals.

The results also support the claims in [10] that the increased performance obtained by FEA—or, in that case, OSI specifically—is due to a representation of each subpopulation that allows communication and competition to occur between overlapping populations. When there is no overlap between subpopulations, such as in the CC algorithms, the performance was significantly worse than when there is overlap, like in FEA. By defining each node's subpopulation to cover its Markov blanket when performing abductive inference, we ensure that each population learns the state assignments for all variables upon which that node may depend. Also, because multiple subpopulations optimize over each variable, the FEA algorithm allows greater exploration of the search space. Through competition and sharing, FEA is able to find good combinations of variable state assignments from the subpopulations are used in the final solution.

While FEA does find better solutions, we observe that FEA almost always requires more fitness evaluations. Some of the additional fitness evaluations do come from the competition step. However, as noted earlier, this only makes up around 10% of the total number of fitness evaluations in the situations we observed. We believe this percentage is related to the number of iterations FEA uses to update each subpopulation. If FEA only ran each subpopulation for one iteration, then the percentage of fitness evaluations used during the competition step would increase. However, if the FEA had each subpopulation run for more iterations, than the percentage of fitness evaluations incurred during competition would decrease.

We believe there are two possible explanations for other additional fitness evaluations. The first is that FEA maintains more diversity between individuals and therefore takes longer to converge than the single populations and CC algorithms. However, when looking at the fitness curves in Figs. 1 and 2, FEA-PSO and FEA-DMVPSO required fewer FEA iterations than the single population algorithms. This is likely to vary based on the algorithm and problem.

Another possible source of the increase in fitness evaluations is the fact that FEA updates each subpopulation for ten iterations. Only after updating each of the subpopulations and performing competition and sharing does FEA check to see if convergence has occurred. If FEA and the single population algorithm both performed the same number of iterations, one would expect FEA to use ten times the number of fitness evaluations. In other words, the single population algorithm would perform 90% fewer fitness evaluations than FEA. Based on our results in Table XI, this number is usually between 20% and 80% in practice. However, in the worst cases, such as FEA-GA and FEA-DE on the discrete problems, the single population algorithm used 97% fewer fitness evaluations than FEA. In these two cases, we believe that the FEA versions maintain greater diversity between the individuals, and therefore require more iterations to converge.

VI. Conclusion

In this paper, we have made several contributions. The first is that we have provided a formal definition for the FEA algorithm. Second, we showed empirically that taking advantage of groupings of highly related variables is the best way to create subswarms. We compared different architectures for several applications and determined that, in the majority of cases, the factor architectures that combined all directly interacting variables in subpopulations performed the best. This paper also demonstrates that FEA's performance is not related to the specific underlying optimization algorithm, through experiments comparing various versions of FEA that use multiple well-known population- and swarm-based algorithms. Additionally, we showed that FEA often outperforms CC methods to a significant degree, reinforcing the importance of the overlapping factors to FEA's success. Finally, we present results regarding the number of fitness evaluations required by FEA versus single-population algorithms, and showed that the competition step in FEA contributes only a small fraction of these additional fitness evaluations.

There are a variety of areas we wish to explore for future work. The first is to explore the convergence properties of FEA further. While FEA appears to always converge to good solutions in practice, the theoretical convergence properties are still under investigation. To accomplish this, we plan to determine if convergence proofs for various EA methods, such as PSO, GA, and CCGA, can be adapted and applied to FEA. A related area of work is the empirical convergence properties of FEA. For example, we would like to investigate how FEA's parameters, such as the number of iterations during interfactor and intrafactor optimization affect FEA's performance.

The complexity of FEA also merits further analysis. As demonstrated in our results, FEA requires more fitness evaluations than its single-population counterparts. However, we believe this is influenced by the number of iterations FEA allows each subpopulation to perform during the solve step. To verify this, we plan to vary the number of iterations allowed during the solve step and compare the performance in terms of fitness and number of fitness evaluations.

Beyond this, we also want to explore hierarchical FEA architectures. Based on the results in Section IV, we see that the size of the factor can influence the performance of the different FEA architectures. We want to explore creating a hierarchical FEA for problems where factor sizes become too large to continue to avoid hitchhiking. In those cases, subfactors could be created, which would result in a single FEA itself being composed of several FEAs.

Another area we want to explore is mapping functions to probabilistic graphical models. If the function is mapped to a probabilistic graphical model, we can then utilize the architectures presented here as way to derive a factor architecture quickly.

Finally, we plan to apply FEA to a wider range of optimization problems; for example, additional benchmark test functions and combinatorial optimization problems such as 3-SAT. This will help inform us further regarding to what type of problems FEA is most effective at solving. Additionally, we want to investigate the scalability of FEA by applying it to large optimization problems.

Acknowledgment

The authors 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 paper. The authors would also like to thank Dr. B. Haberman at the Johns Hopkins University Applied Physics Laboratory and Dr. K. G. Pillai at Cerner Corporation for their ideas during the formative stages of this research.

REFERENCES

[1] J. C. Spall, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. New York, NY, USA: Wiley, 2005.
[2] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI, USA: Univ. Michigan Press, 1975.
[3] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 4–31, Feb. 2011.
[4] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, Perth, WA, Australia, 1995, pp. 1942–1948.
[5] M. Mitchell, S. Forrest, and J. H. Holland, "The royal road for genetic algorithms: Fitness landscapes and GA performance," in *Proc. 1st Eur. Conf. Artif. Life*, Cambridge, MA, USA, 1992, pp. 245–254.
[6] Y.-J. Shi, H.-F. Teng, and Z.-Q. Li, "Cooperative co-evolutionary differential evolution for function optimization," in *Advances in Natural Computation*. Heidelberg, Germany: Springer, 2005, pp. 1080–1088.
[7] F. Van den Bergh and A. P. Engelbrecht, "Cooperative learning in neural networks using particle swarm optimizers," *South African Comput. J.*, vol. 26, pp. 84–90, 2000.
[8] S. Cheng, Y. Shi, and Q. Qin, "Population diversity of particle swarm optimizer solving single-and multi-objective problems," *Int. J. Swarm Intell. Res.*, vol. 3, no. 4, pp. 23–60, 2012.
[9] K. E. Parsopoulos and M. N. Vrahatis, "On the computation of all global minimizers through particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 211–224, Jun. 2004.
[10] N. Fortier, J. Sheppard, and S. Strasser, "Abductive inference in Bayesian networks using distributed overlapping swarm intelligence," *Soft Comput.*, vol. 19, no. 4, pp. 981–1001, 2015.
[11] K. G. Pillai and J. Sheppard, "Overlapping swarm intelligence for training artificial neural networks," in *Proc. IEEE Swarm Intell. Symp. (SIS)*, Paris, France, 2011, pp. 1–8.
[12] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel Problem Solving From Nature—PPSN III*. Heidelberg, Germany: Springer, 1994, pp. 249–257.
[13] M. A. Potter and K. A. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evol. Comput.*, vol. 8, no. 1, pp. 1–29, 2000.
[14] F. Van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 225–239, Jun. 2004.
[15] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Inf. Sci.*, vol. 178, no. 15, pp. 2985–2999, 2008.
[16] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 16, no. 2, pp. 210–224, Apr. 2012.
[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[18] B. K. Haberman and J. W. Sheppard, "Overlapping particle swarms for energy-efficient routing in sensor networks," *Wireless Netw.*, vol. 18, no. 4, pp. 351–363, 2012.
[19] N. Fortier, J. W. Sheppard, and K. G. Pillai, "DOSI: Training artificial neural networks using overlapping swarm intelligence with local credit assignment," in *Proc. Joint IEEE 6th Int. Conf. Soft Comput. Intell. Syst. (SCIS) 13th Int. Symp. Adv. Intell. Syst. (ISIS)*, Kobe, Japan, 2012, pp. 1420–1425.
[20] N. Fortier, J. Sheppard, and K. G. Pillai, "Bayesian abductive inference using overlapping swarm intelligence," in *Proc. IEEE Swarm Intell. Symp. (SIS)*, Singapore, 2013, pp. 263–270.
[21] N. Fortier, J. Sheppard, and S. Strasser, "Learning Bayesian classifiers using overlapping swarm intelligence," in *Proc. IEEE Swarm Intell. Symp. (SIS)*, Orlando, FL, USA, 2014, pp. 1–8.
[22] N. Fortier, J. Sheppard, and S. Strasser, "Parameter estimation in Bayesian networks using overlapping swarm intelligence," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*. Madrid, Spain, 2015, pp. 9–16.
[23] A. Gupta, Y.-S. Ong, and L. Feng, "Multifactorial evolution: Toward evolutionary multitasking," *IEEE Trans. Evol. Comput.*, vol. 20, no. 3, pp. 343–357, Jun. 2016.
[24] K. Veeramachaneni, L. Osadciw, and G. Kamath, "Probabilistically driven particle swarms for optimization of multi valued discrete problems: Design and analysis," in *Proc. IEEE Swarm Intell. Symp. (SIS)*, Honolulu, HI, USA, 2007, pp. 141–149.
[25] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann, 1988.
[26] M. Scutari. (2012). *Bayesian Network Repository*. [Online]. Available: http://www.bnlearn.com/bnrepository/
[27] H. Aguirre and K. Tanaka, "A study on the behavior of genetic algorithms on NK-landscapes: Effects of selection, drift, mutation, and recombination," *IEICE Trans. Fundamentals Electron. Commun. Comput. Sci.*, vol. E86-A, no. 9, pp. 2270–2279, 2003.
[28] A. P. Engelbrecht, "Fitness function evaluations: A fair stopping condition?" in *Proc. IEEE Swarm Intell. Symp. (SIS)*, Orlando, FL, USA, 2014, pp. 1–8.
[29] J. Lampinen and I. Zelinka, "Mixed integer-discrete-continuous optimization by differential evolution," in *Proc. 5th Int. Conf. Soft Comput.*, 1999, pp. 77–81.

Shane Strasser (M'11) received the B.S. degree in computer science and mathematics from the University of Sioux Falls, Sioux Falls, SD, USA, and the M.S. degree in computer science from Montana State University, Bozeman, MT, USA, where he is currently pursuing the Ph.D. degree in computer science.

His current research interests include artificial intelligence and machine learning with a focus on evolutionary and swarm algorithms.

Mr. Strasser has received several awards, including the 2012 Outstanding Ph.D. MSU Computer Science Researcher Award and the 2011 AUTOTESTCON Best Student Paper Award.

John Sheppard (F'07) received the B.S. degree in computer science from Southern Methodist University, Dallas, TX, USA, and the M.S. and Ph.D. degrees in computer science from Johns Hopkins University, Baltimore, MD, USA.

He was the Inaugural RightNow Technologies Distinguished Professor in Computer Science with Montana State University, Bozeman, MT, USA, where he currently holds an appointment as the College of Engineering Distinguished Professor with the Computer Science Department, MSU. He is also an Adjunct Professor with the Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA. He was a Fellow with ARINC Inc., Annapolis, MD, USA, for almost 20 years. His current research interests include Bayesian classification, dynamic Bayesian networks, evolutionary methods, and reinforcement learning.

Nathan Fortier (M'10) received the undergraduate degree in computer science from Montana Tech, Butte, MT, USA, in 2011, and the M.S. and Ph.D. degrees from Montana State University, Bozeman, MD, USA, in 2013 and 2015, respectively.

He is currently an Applications Engineer with Golden Helix, Bozeman. His current research interests include swarm algorithms and probabilistic graphical models.

Dr. Fortier was a recipient of the MSU's Award for Outstanding Ph.D. Computer Science Researcher in 2015.

Rollie Goodman (M'16) received the undergraduate degree in economics from Lewis & Clark College, Portland, OR, USA, in 2014. She is currently pursuing the M.S. degree in computer science with Montana State University, Bozeman, MD, USA, focusing on artificial intelligence and machine learning.

Her current research interests include evolutionary and swarm-based algorithms.