Relaxing Consensus in Distributed Factored Evolutionary Algorithms

Stephyn Butcher
Johns Hopkins University
Dept. of Computer Science
Baltimore, MD 21218 -2682
steve.butcher@jhu.edu

Shane Strasser
Montana State University
Dept. of Computer Science
Bozeman, MT 59717-3880
shane.strasser@msu.montana.edu

Jenna Hoole
Whitworth University
Dept. of Mathematics and Computer Science
Spokane, WA 99251
jhoole15@my.whitworth.edu

Benjamin Demeo
Williams College
Dept. of Computer Science
Williamstown, MA 01267
bd2@williams.edu

John Sheppard
Montana State University
Dept. of Computer Science
Bozeman, MT 59717-3880
john.sheppard@msu.montana.edu

ABSTRACT

Factored Evolutionary Algorithms (FEA) have proven to be fast and efficient optimization methods, often outperforming established methods using single populations. One restriction to FEA is that it requires a central communication point between all of the factors, making FEA difficult to use in completely distributed settings. The Distributed Factored Evolutionary Algorithm (DFEA) relaxes this requirement on central communication by having neighboring factors communicate directly with one another. While DFEA has been effective at finding good solutions, there is often an increase in computational complexity due to the communication between factors. In previous work on DFEA, the authors required the algorithm reach full consensus between factors during communication. In this paper, we demonstrate that even without full consensus, the performance of DFEA was not statistically different on problems with low epistasis. Additionally, we found that there is a relationship between the convergence of consensus between factors and the convergence of fitness of DFEA.

1. INTRODUCTION

Overlapping Swarm Intelligence (OSI) is a swarm based algorithm that has been found to produce high quality solutions on a wide range of problems. OSI differs from basic PSO in that it creates subswarms, or factors, that optimize over overlapping subcomponents of the full solution. Similar to how factorization in mathematics decomposes a polynomial into a product of factors, OSI decomposes the optimization problem into a set of factors that, when put together, represent full solutions to the problem. OSI has been applied successfully to a wide range of problems, such as energy aware routing in sensor networks [7], training deep neural networks [14], performing abductive inference in Bayesian networks [3], and learning Bayesian networks [4, 5]. A generalization of OSI, called Factored Evolutionary Algorithms (FEA), allows for any evolutionary algorithm to be used as the underlying optimization algorithm [16]. While OSI requires the factors overlap with one another [3], FEA does not, which allows the FEA framework to include multi-population algorithms such as Cooperative Co-evolutionary Genetic Algorithms [16].

One of the downsides to FEA is that making the algorithm completely distributed is difficult because FEA relies on a single full global solution to be communicated between all of the factors. There has been work on developing distributed versions of OSI, called Distributed Overlapping Swarm Intelligence (DOSI), in which the algorithm no longer requires a single full global solution to be maintained between all factors [3, 6]. Instead, each factor in DOSI maintains its own full solution that is updated during a sharing step. While this allows for DOSI to be completely distributed, it can increase the runtime. This is because all previous work on DOSI required factors' full solutions to reach full consensus during sharing [3]. Depending on the problem and factor architecture, this can be computationally expensive.

In this paper, we provide a generalization of DOSI, called Distributed Factored Evolutionary Algorithm, that is similar to FEA's generalization of OSI. This allows for DFEA to use any optimization algorithm as the underlying optimization process. Next, we investigate the effect that relaxing the amount of consensus between factors has on DFEA's performance. We hypothesize that there is a relationship between the amount of consensus required during the sharing step in the DFEA and the degree of epitasis in the problem and that this relationship affects the solution quality. This hypothesis was tested on various versions of DFEA performing abductive inference in Bayesian networks, maximizing NK landscapes, and minimizing benchmark test functions. Each version of DFEA reduces the number of sharing iterations during the share step, thereby reducing the amount of consensus between factors. We discover that during each individual sharing step, full consensus between the factors' full

GECCO '16, July 20-24, 2016, Denver, CO, USA

DOI: http://dx.doi.org/10.1145/2908812.2908936

personal solutions is not required. However, a certain degree of consensus is required in order to obtain quality solutions. This degree of consensus is controlled by the number of sharing steps. In addition, we present results that show that a relationship exists between factors reaching consensus and the convergence of fitness in DFEA.

2. RELATED WORK

Overlapping Swarm Intelligence (OSI) is a swarm based algorithm based on Particle Swarm Optimization (PSO). The first version of OSI was introduced in 2012 by Haberman and Sheppard [7]. In their paper, the authors created an algorithm called "Particle-based Routing with Overlapping Swarms for Energy Efficiency" (PROSE) that works by creating multiple swarms that are assigned to overlapping subproblems. PROSE was 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. The algorithm found solutions that were able to extend the life of the sensor networks and to perform significantly better than current energy-aware routing protocols [7].

PROSE was then adapted by Ganesan Pillai and Sheppard to learn the weights of deep artificial neural networks [14]. It was in this paper that OSI was introduced, where each swarm represents a unique path starting at an input node and ending at each output node. Thus the set of swarms corresponds to all paths through the network, and each swarm then learns the weights for that path. A common vector of weights, called the full global solution, is 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. In their work, the authors show that OSI outperformed several other PSO based algorithms as well as standard backpropagation on deep networks [14].

OSI has 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.* applied OSI to perform full and partial abductive inference in Bayesian networks where a subswarm is created for each node in the network, encoding the states of that node and its Markov blanket [2, 3]. The authors were able to show that OSI outperformed several other population-based and traditional algorithms, such as PSO, GA, simulated annealing, stochastic local search, and mini-bucket elimination [3].

Another application of OSI includes learning Bayesian networks. Fortier *et al.* 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 [4]. For each variable in the network, two subswarms were created: one for the incoming links and one for the outgoing links. The authors were able to show that in most cases OSI was able to outperform the competing approaches significantly. Additionally, the work included the earliest complexity results on OSI.

When learning Bayesian networks, latent or unobserved variables are often introduced into the network to reduce the number of parameters required by the network or to introduce new dependencies between other variables. Fortier *et al.* used OSI to learn the parameters of latent variables in Bayesian networks [5]. A subswarm was created for each node with unlearned parameters and all of the variables in that node's Markov blanket. The authors were able to show that OSI outperformed PSO and Expectation-Maximization variants and that the amount of overlap between the subswarms can impact the performance of OSI.

Fortier adapted OSI to perform general structure learning of Bayesian network [1]. In that work, each subswarm learned the the links for its Markov blanket and was found to outperform traditional Bayesian learning algorithms.

DOSI was first developed by Fortier *et al.* to learn weights on deep neural networks [6]. The key distinction from OSI is that a full global solution is not used for fitness evaluation. Instead, each subswarm maintains its own full personal solution, which allows for the algorithm to be distributed more effectively. A communication and sharing algorithm was defined so that subswarms could share values while also competing with one another. The authors 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.

Similar to OSI, DOSI has been adapted to perform full and partial abductive inference in Bayesian networks. DOSI was found to be comparable to OSI on most problems and was only outperformed on large Bayesian networks or when the explanation sets are greater than 4 [3]. The authors also demonstrated that DOSI required more fitness evaluations than OSI [3].

FEA was first introduced by Strasser *et al.*, which generalizes OSI so that any evolutionary algorithm can be used as the underlying optimization technique [16]. In addition, the authors were able to show the performance of FEA is dependent on its factor architecture. This factor architecture is a decomposition of the optimization problem into subproblems that is based on the factor functions of the fitness function's factor graph. The paper demonstrated that when there is a low number of interactions between variables, the optimal way to derive factors is by using the variables' Markov blanket. However, when there are a high number of interactions, FEA performs better when factors are derived by using a variables' factor function. Finally, the authors demonstrate that FEA still performs well even when using simple optimization algorithms, such as Hill Climbing [16].

3. BACKGROUND

Our experiments with the DFEA require a component optimization algorithm and a set of test problems. For this particular application of DFEA, we utilize Discrete Multi-Value Particle Swarm Optimization (DMVPSO) proposed by Veeramchaneni *et al.* as the underlying optimization algorithm [18].

For the test problems we chose NK landscapes, abductive inference in Bayesian Networks and some common benchmark optimization problems. NK landscapes were included because they represent commonly used functions for evaluating the performance of evolutionary and swarm algorithms. We included abductive inference in Bayesian Networks because they are a practical application of optimization. Additionally, Fortier *et al.* showed that OSI outperforms domain specific algorithms like approximate mini-bucket elimination on complex networks [3].

3.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a swarm based approach developed by Kennedy and Eberhart to optimize a function, usually over a continuous set of variables, based on the behavior of fish schools and bird flocks [8]. Given a function $f : \mathbb{R}^n \to \mathbb{R}$ to be optimized with parameters $\mathbf{X} = \langle X_1, X_2, \ldots, X_n \rangle$, PSO uses a set of particles where each particle p_i contains a vector $\mathbf{X}_i = \langle X_{i,1}, X_{i,2}, \ldots, X_{i,n} \rangle$ that represents a candidate solution. This is the particle's current position in the search space. In addition, a particle uses a velocity vector $\mathbf{V}_i$ that controls the particle's movement. Each particle keeps track of its own best position found in a vector, $\mathbf{pBest}_i$ and the best position discovered by the entire swarm, **gBest**. During each iteration, a particle's position is updated as follows:

$$\mathbf{V}_i = \omega \mathbf{V}_i + U(0, \phi_1) \otimes (\mathbf{pBest}_i - \mathbf{X}_i) + U(0, \phi_2) \otimes (\mathbf{gBest} - \mathbf{X}_i) \quad (1)$$

$$\mathbf{X}_i = \mathbf{X}_i + \mathbf{V}_i \quad (2)$$

where $U(0, \phi_1)$ and $U(0, \phi_2)$ are random numbers distributed uniformly between 0 and ϕ_1 and ϕ_2 respectively, and cause the particles to have more random paths, which aid in the exploration of the search space. The symbol $\otimes$ represents component-wise multiplication, and ω is an inertia value that helps the velocity values from growing out of control.

While the PSO equations shown in Equations (1) and (2) have been shown to work well on optimization problems involving continuous variables, many real-world problems operate over a set of discrete variables. Several extensions to the standard PSO algorithm have been developed that enable optimizing functions over discrete states. Veeramachaneni *et al.* presented an algorithm that allows PSO to optimize discrete multi-valued functions called Discrete Multi-Valued PSO (DMVPSO) [18]. In this algorithm, the velocity update equations remain mostly unchanged. However, the semantics of the velocity vector are changed to denote the probability of a particle's position term having a value $[0, M-1]$. The update to the position vector is also modified to take advantage of the new velocity vector semantics. Each dimension in the velocity vector is restricted to values in $[0, M-1]$, where M is the cardinality of the dimension. After the velocity is updated, it is mapped into a $[0, M-1]$ interval using the sigmoid function

$$S_{i,j} = \frac{M-1}{1 + \exp(-V_{i,j})}.$$

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. Finally, the result is passed through the piecewise function

$$X_{i,j} = \begin{cases} M-1 & X_{i,j} > M-1 \\ 0 & X_{i,j} < 0 \\ X_{i,j} & \text{otherwise} \end{cases}$$

to ensure the values remains in the range $[0, M-1]$.

3.2 The NK Model

The NK landscape is a mathematical framework that generates tunable fitness landscapes that are often used as test functions for evaluating EAs [11]. An NK landscape model contains two parameters, N and K, that control the overall size of the landscape and the structure or amount of interaction between each dimension, respectively [10].

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

There are multiple ways to define the neighborhood function. The simplest way is to 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. In other cases, the neighborhood of each bit is created randomly.

3.3 Bayesian Networks

A Bayesian network is a directed acyclic graph $G = (\mathbf{V}, \mathbf{E})$ that encodes a joint probability distribution over a set of random variables, where each variable can assume one of an arbitrary number of mutually exclusive values [12, 13]. In a Bayesian network, each random variable X_i is represented by a node, and edges between nodes in the network represent probabilistic relationships between the random variables. Each root node contains a prior probability distribution while each non-root node contains a probability distribution conditioned on the node's parents.

For any set of random variables in the network, the joint probability distribution can be represented using the local distributions in the network

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

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

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

A common type of query for Bayesian networks is the probability distribution of a variable given a set of evidence. Another type of query is called abductive inference, which finds the most probable state assignment $\mathbf{x}$ to the variables in $\mathbf{X}_U$ given the evidence $\mathbf{X}_O = \mathbf{x}_O$. This is also known as the Maximum *A Priori* (MAP) probability state of the variables of a network. In addition, users often ask for the top k hypotheses. When $k > 1$, this is often referred to as the k-Most Probable Explanation (k-MPE) problem.

4. DISTRIBUTED FEA

DFEA is an extension of FEA that allows the algorithm to be distributed completely. In FEA, all of the factors use a full central solution $\boldsymbol{G}$ to evaluate the solutions. DFEA breaks the dependency of all factors requiring access to a central solution by having each factor maintain its own full personal solution. However, this requires certain portions of the FEA algorithm to be adapted to allow for competition and sharing between neighboring factors.

FEA takes a function $f : \mathbb{R}^n \to \mathbb{R}$ with parameters $\mathbf{X} = \langle X_1, X_2, \ldots, X_n \rangle$ and creates a set $\boldsymbol{\mathcal{S}}$ of s subpopulations, or factors. Each $\mathbf{S}_i \in \boldsymbol{\mathcal{S}}$ is a subsequence of $\mathbf{X}$ of size k.

Note that f can still be optimized over the variables in $\mathbf{S}_i$ by holding variables $\mathbf{R}_i = \mathbf{X} \setminus \mathbf{S}_i$ constant. 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 $s > 1$, $\mathbf{S}_i = \mathbf{X}$ for all factors, and $\bigcup \mathbf{S}_i = \mathbf{X}$ for all populations, the algorithm becomes a multi-population algorithm.

FEA is the case where there are factors that are proper subsets of $\mathbf{X}$ that may or may not overlap with one another. In this work, we look at problems where every factor overlaps with some other factor. Should there be a disjoint factor, we have a family of FEA and DFEAs.

In FEA, all of the factors' remainder variables $\boldsymbol{R}_i$ are guaranteed to have the same state assignments because they were filled with a full global solution $\boldsymbol{G}$. However, in DFEA, the remainder variables $\boldsymbol{R}_i$ are not set by $\boldsymbol{G}$ and are instead assigned by using $\boldsymbol{S}_i$'s neighbors. We now describe the sharing and competition algorithms that update the factors' full personal solutions $\boldsymbol{R}_i \cup \boldsymbol{S}_i$.

4.1 Competition

The purpose of the competition phase in DFEA is to determine which factor has the best value for every dimension and to resolve any conflicts neighboring factors may have about a variable's value. In FEA, competition is held by the full global solution $\boldsymbol{G}$. However, because DFEA does not have a full global solution $\boldsymbol{G}$, competition is held by each individual factor that acts as the arbiter of which value from competing factors should be communicated to the remaining factors. Here we present a general DFEA competition algorithm based on the work done by Fortier *et al.* [3].

First, we must define an *arbiter* node for dimension X_i to be a factor that performs the competition for the variable X_i. The arbiter's full personal solution is then used to evaluate other values during competition. Each arbiter node for X_i communicates directly with any factor $\mathbf{S}_j$ that contains X_i, inducing a communication topology between the factors. We define this induced graph H as the DFEA's *communication graph*, were the nodes represent factors. An edge connects two nodes in H if and only if one of the nodes is an arbiter for a value that the other node also optimizes over. Note that two factors $\mathbf{S}_i$ and $\mathbf{S}_j$ can overlap with one another but not communicate directly with one another. This occurs if $\mathbf{S}_i$ is the arbiter for variable X_i and $X_i \notin \mathbf{S}_j$. The compete algorithm for DFEA is shown in Algorithm 1.

The Compete Algorithm works as follows. First, the function "arbiter" in line 2 returns the index of the factor that is the arbiter for dimension X_i. After initializing the comparison variables in lines 3-4, the algorithm iterates over a random permutation of all factors optimizing X_i, denoted $\mathcal{S}_i$. This random ordering is generated each time the algorithm is called. In line 6, the value from the factor is substituted into the arbiter's solution and then compared to the current best value (lines 7 - 10). The value that results in the best fitness is then returned by the algorithm.

DFEA's compete algorithm only relies on the arbiter factor for X_i and the factors that also optimize X_i. Note that the competition algorithm is not guaranteed to find the best combination of values from the factors. However, by iterating over random permutations of $\mathcal{S}_i$, the algorithm is able to explore different combinations and is still able to find good combinations of values.

Algorithm 1 DFEA Competition Algorithm

Input: Factors $\mathcal{S}$, Function f

1: **for** $X_i \in \boldsymbol{X}$ **do**
2: $a \leftarrow$ arbiter(X_i)
3: $bestFit \leftarrow \infty$
4: $bestVal \leftarrow \boldsymbol{S}_a[X_i]$
5: **for all** $\boldsymbol{S}_j \in$ Optimizers(X_i) **do**
6: $\boldsymbol{S}_a[X_i] \leftarrow \boldsymbol{S}_j[X_i]$
7: **if** $f(\boldsymbol{S}_a \cup \boldsymbol{R}_a)$ is better than $bestFit$ **then**
8: $bestVal \leftarrow \boldsymbol{S}_j[X_i]$
9: $bestFit \leftarrow f(\boldsymbol{S}_a \cup \boldsymbol{R}_a)$
10: **end if**
11: **end for**
12: **for** $\boldsymbol{S}_j \in$ Optimizers(X_i) **do**
13: $\boldsymbol{S}_j.\boldsymbol{G}[X_i] \leftarrow bestVal$
14: **end for**
15: **end for**

Algorithm 2 DFEA Share Algorithm

Input: Factors $\mathcal{S}$

1: **for** $k = 1$ to C **do**
2: **for all** $\boldsymbol{S}_i \in \mathcal{S}$ and $\boldsymbol{S}_j \in$ neighbors$(\boldsymbol{S}_i)$ **do**
3: Exchange$(\boldsymbol{S}_i, \boldsymbol{S}_j)$
4: **end for**
5: **end for**
6: **return**

4.2 Sharing

In the DFEA Algorithm, the share step's purpose is to let factors distribute information to one another. This is accomplished by having two neighboring factors in the communication graph H exchange information about their full personal solutions. However, neighboring factors need to be augmented with additional information in order for the factors to know which values to share with one another. This is accomplished by having each factor maintains a δ-map. For each dimension X_i, the δ-map stores the minimum number of steps required to reach a factor learning X_i, where a step can occur only between neighboring factors. For example, if factor $\boldsymbol{S}_i$ learns dimension X_j, then $\boldsymbol{S}_i.\delta_j = 0$. If $\boldsymbol{S}_i$ does not learn dimension X_j but neighbors a factor that does, then $\boldsymbol{S}_i.\delta_j = 1$.

Let $d_H(\boldsymbol{S}_i, \boldsymbol{S}_j)$ denote the distance or the minimum number of hops between nodes corresponding to $\boldsymbol{S}_i$ and $\boldsymbol{S}_j$ in the graph H. Then $\boldsymbol{S}_i.\delta_k = \min\{d_G(\boldsymbol{S}_i, \boldsymbol{S}_j) | \boldsymbol{S}_j \text{ learns } X_k\}$. We say that the factors reach *consensus* when they all agree on all state assignments.

Initially, $\boldsymbol{S}_i.\delta_j = 0$ if $\boldsymbol{S}_i$ optimizes X_j and $\boldsymbol{S}_i.\delta_j = \infty$ otherwise. We say that factor $\boldsymbol{S}_i$ knows dimension X_k once $\boldsymbol{S}_i.\delta_k < \infty$. The full share and exchange algorithms for DFEA are shown in Algorithm 2 and 3, respectively.

The Share algorithm operates as follows. For C iterations, the algorithm iterates over all neighboring pairs of factors in H and calls the exchange algorithm for those two factors.

During the Exchange algorithm, all of the n dimensions in $\boldsymbol{X}$ are iterated over. In lines 2-5, the algorithm compares the δ values of the two factors for the current dimension k. If the δ_k value for factors $\boldsymbol{S}_j$ is lower than δ_k from $\boldsymbol{S}_i$, then the value from $\boldsymbol{S}_j$ is inserted into $\boldsymbol{S}_i$. In addition, δ_k for $\boldsymbol{S}_i$ is updated according to line 3. Lines 5-8 do the same thing except that information is shared from S_i to S_j. Note that

Algorithm 3 DFEA Exchange Algorithm

Input: Factors $\boldsymbol{S}_i$, $\boldsymbol{S}_j$

1: **for** $k = 1$ to n **do**
2: **if** $\boldsymbol{S}_i.\delta_k > \boldsymbol{S}_j.\delta_k$ **then**
3: $\boldsymbol{S}_i.\delta_k \leftarrow \boldsymbol{S}_j.\delta_k + 1$
4: $\boldsymbol{S}_i.\boldsymbol{G}[k] \leftarrow \boldsymbol{S}_j.\boldsymbol{G}[k]$
5: **else if** $\boldsymbol{S}_i.\delta_k < \boldsymbol{S}_j.\delta_k$ **then**
6: $\boldsymbol{S}_j.\delta_k \leftarrow \boldsymbol{S}_i.\delta_k + 1$
7: $\boldsymbol{S}_j.\boldsymbol{G}[k] \leftarrow \boldsymbol{S}_i.\boldsymbol{G}[k]$
8: **end if**
9: **end for**
10: **return**

Algorithm 4 Distributed Factored Evolutionary Algorithm

Input: Function f
Output: Solution $\boldsymbol{G}$

1: $\mathcal{S} \leftarrow$ initalizeFactors$(f, \boldsymbol{X})$
2: **repeat**
3: **for** $\boldsymbol{S}_i \in \mathcal{S}$ **do**
4: $\boldsymbol{S}_i$.optimize(f)
5: **end for**
6: Compete$(\mathcal{S})$
7: Share$(\mathcal{S})$
8: **until** Termination Criterion is Met
9: **return** $\boldsymbol{G}$

$\boldsymbol{S}_i.\boldsymbol{G}[\ ]$ is the factor's full personal solution and is equal to $\boldsymbol{S}_i \bigcup \boldsymbol{R}_i$.

4.3 Algorithm

Now that Compete and Share algorithms have been defined for DFEA, we present the DFEA algorithm (Algorithm 4). DFEA is initialized similarly to FEA but does not require initializing full global solution $\boldsymbol{G}$. Instead each factor $\boldsymbol{S}_i$ contains its full personal solution that is also initialized in line 1. After the intra-population optimization in lines 3 - 5, the algorithm holds a competition by calling the Compete function (line 6). Finally, the algorithm performs sharing in line 7. This process is repeated until the termination criteria is met.

5. EXPERIMENTS

One of the defining differences between FEA and DFEA is the necessity of the Share step. Each subswarm in DFEA has a local rather than central view of a current best-so-far solution. The Share step is thus necessary to exchange information between subswarms in order for them to reach consensus. Depending on the size and complexity of the communication topology, this can increase the runtime of DFEA significantly. In order to reduce this runtime, we investigate the effect that relaxing the amount of consensus between factors has on DFEA's performance. To test this, we relax the number of sharing iterations C that are used in the DFEA algorithm, which in turn causes the subpopulations to reach a lower level of consensus. We also applied DFEA and consensus relaxation to several general optimization problems which had not been done before.

5.1 Design

To test our hypothesis, we created three different versions of DFEA: DFEA-1, DFEA-1/2, and DFEA-Full. DFEA-1 used only 1 sharing iteration during the Share algorithm while DFEA-1/2 used $Round(D/2)$ sharing iterations, where D is the diameter of graph induced by the communication topology. DFEA-Full ran D sharing iterations. We also ran the FEA algorithm for an additional comparison. All FEA and DFEA versions ran for 10 inter-swarm optimization iterations since this value was found by Strasser *et al.* [16] to allow FEA to converge.

For our experiments we used three sets of problems: maximizing NK landscapes, performing abductive inference on Bayesian networks, and optimizing several standard benchmark functions.

For the NK landscapes and abductive inference, FEA and DFEA used DMVPSO as the underlying optimization algorithm. On the benchmark problems, we used canonical PSO. For both PSOs, the ω parameter was set to 0.729, and ϕ_1 and ϕ_2 were both set to 1.49618.

We applied these same versions of PSO, FEA and DFEA as well as the relaxed versions of DFEA to the benchmark optimization problems. The individual and component PSO parameters were the same. The individual PSO was run for 100 iterations with population sizes of 10 times the dimensions. The FEA and DFEA were run for 20 compete-and-share iterations with the component PSOs running for 5 iterations. This gives single swarm algorithms the same total number of iterations as the factors in DFEA. Each factor for FEA and DFEA had a population size of 10, which gives FEA and DFEA the same number of individuals as the single swarm algorithms.

5.1.1 NK Landscapes

We generated NK landscapes with parameters $N = 25$ and 40 and $K = 2, 5$ and 10. For each set of parameters, we created 30 random landscapes.

In applying DFEA to NK landscapes, we used the Neighborhood architecture proposed by Strasser *et al.* [16] since it outperformed competing factor architecture approaches. The Neighborhood architecture creates a factor for each variable X_i and adds to the factor variable X_i and all variables in the set $nb_K(X_i)$. This results in factors of size $K + 1$.

5.1.2 Bayesian Networks

For abductive inference on Bayesian networks, we used the Hailfinder, Hepar2, Insurance, and Win95pts Bayesian networks from the Bayesian Network Repository [15]. These networks were chosen to be consistent with [3].

To evaluate the fitness of a state assignment we used the log likelihood ℓ, which is calculated as

$$\ell(\mathbf{x}) = \sum_{i=1}^{n} \log P(x_i|\mathrm{Pa}(x_i))$$

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

The factor architecture chosen was the Markov architecture proposed by Fortier *et al.* since this was shown to outperform all other architectures on Bayesian networks [3, 16]. This uses the Markov blanket of every node to create subpopulations, because it offers one of the most natural ways to subdivide a Bayesian network and provide overlap. Additionally, it gives the algorithm an advantage because every node in the network is conditionally independent of all other nodes when conditioned on its Markov blanket. For our ex-

periments we used an empty evidence set to keep results comparable .

5.1.3 Benchmark Optimization Problems

We picked a variety of benchmark optimization problems: Sphere, Exponential, Schwefel 1.2, Dixon-Price, Ackley's, Rosenbrock, and Griewank [9]. All of the problems are minimization problems with global minima at 0.0 except for the Exponential which has a minimum at -1.0. All of the problems are scalable, meaning they can be optimized for versions of any dimension. The Sphere function is separable. The remaining functions are non-separable with most functions depending on adjacent, overlapping dimensions such as x_i and x_{i+1}. Because of this, we used a factor size of two for all of the benchmark optimization problems.

5.2 Results

Table 1 shows the results comparing FEA and all versions of DFEA on performing abductive inference on Bayesian networks and maximizing NK landscapes. Note that these problems are maximization. Results comparing PSO, FEA, and DFEA on minimizing the benchmark functions are in Table 2 while the results comparing the different versions of DFEA on the benchmark functions are in Table 3. All results are expressed as means over 30 trials with standard errors in parentheses.

In the Bayesian network problems, there are only small differences between all versions of DFEA. In some most cases, DFEA-Full performs the best. DFEA-1/2 performs better than DFEA-1 and DFEA-Full only on the Win95pts network. In the Insurance network, DFEA-1 performs better than DFEA-Full, but only by a small margin. On all networks, all DFEA algorithms are competitive and sometimes better than the FEA algorithm.

For the NK-landscape results, DFEA-1 almost always performed worse than the other DFEA algorithms. In some landscapes, such as $N = 25$ and $K = 2$, DFEA-1/2 performs better than DFEA-Full but for when $N = 25$ and $K = 10$, DFEA-1/2 performs slightly worse than DFEA-Full. When $N = 40$ and $K = 2, 10$, DFEA-1/2 outperforms DFEA-Full, but when $N = 40$ and $K = 5$, DFEA-Full outperforms DFEA-1/2.

On the benchmark optimization problems, DFEA-Full outperformed the PSO except for the Schwefel 1.2 function. Overall, DFEA was slightly worse than FEA on all the benchmark problems. When looking at the consensus results in Table 3 relaxation results are presented. DFEA-Full performed better that DFEA-1 and DFEA-1/2 on Sphere, Exponential, Dixon-Price, and Ackley's. However, DFEA-1 performed the best on Rosenbrock while DFEA-1/2 performed the best was on Griewank. DFEA-1/2 outperformed DFEA-1 on all functions except for Schwefel and Rosenbrock, but was outperformed by DFEA-Full except on Schwefel and Griewank.

6. DISCUSSION

Based on the Bayesian network results, we can see that DFEA does not need to reach full consensus during each Share step in order to find quality solutions. To investigate this, we looked at the the average Hamming distance between DFEA's factors on the Hailfinder Bayesian network (Figure 1). We also looked at fitness curves (Figure 2). In Figure 1, the major X-axis on the chart is the inter-factor

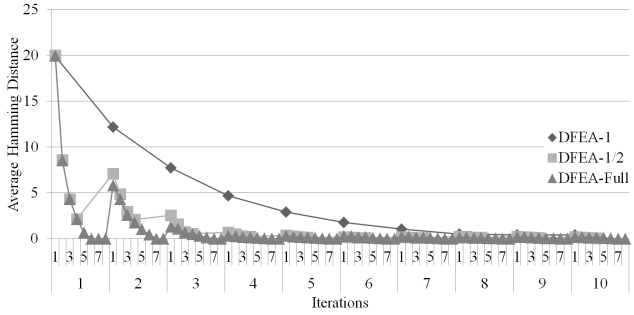

Figure 1: Average consensus between factors over time of DFEA performing abductive inference on the Hailfinder Network.

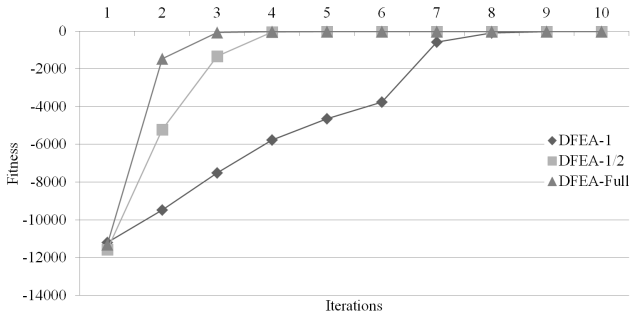

Figure 2: Fitness over time of DFEA performing abductive inference on Hailfinder Network.

optimization iterations while the minor X-axis is the number of sharing iterations for the different DFEA versions. The X-axis in Figure 2 is the inter-factor iteration.

Based on the charts, one can see that in DFEA-1 and DFEA-1/2, the factors are still able to reach consensus over the lifetime of the algorithms because they all eventually reach a Hamming distance of zero. We believe this because when optimizers start converging in their search spaces, the number of values changed during the exchange step decreases and therefore, the factors are able to reach consensus over several DFEA iterations.

We performed a similar analysis for NK landscapes $N = 25$ and $K = 10$. However, we set the number of inter-factor iterations to 50. Figures 3 and 4 show the consensus and fitness graphs for NK landscapes $N = 25$ and $K = 10$.

For NK landscapes $N = 25$ and $K = 10$, DFEA-1 reaches consensus at a much slower rate than DFEA-1/2 and DFEA-Full. Meanwhile, DFEA-1/2 reaches consensus and fitness at about the same rate as DFEA-Full. DFEA-1 may be able to reach the same fitness as DFEA-1/2 and DFEA-Full in more iterations, but the cost of needing more iterations greatly outweighs the reduction in runtime by only having 1 sharing step. This appears to be the case where there is high epistasis in the problems, like on NK landscapes when $K = 5, 10$. When there is high epistasis, the solve and competition steps increase the differences between factors. This necessitates the need for more sharing iterations in order to reduce the difference. When there is low epistasis, the increase in the factors' difference during the solve and com-

Table 1: Results from varying the amount of consensus between factors.

			Diameter	PSO	FEA	DFEA-1	DFEA-1/2	DFEA-Full
Bayesian	Hailfinder		8	−86.94 (23.70)	−33.85 (0.46)	−38.70 (0.49)	−38.58 (0.58)	−36.86 (0.66)
	Hepar2		5	−49.47 (0.45)	−16.85 (0.34)	−20.05 (1.13)	−21.03 (1.01)	−19.62 (1.16)
	Insurance		5	−23.83 (0.24)	−11.22 (0.29)	−12.76 (0.59)	−13.75 (0.48)	−12.78 (0.44)
	Win95pts		5	−90.41 (1.65)	−16.41 (1.34)	−29.86 (1.28)	−28.50 (2.00)	−31.16 (2.07)
NKs	$N = 25$	$K = 2$	12	17.96 (0.02)	18.56 (0.09)	17.69 (0.09)	18.06 (0.10)	18.00 (0.09)
		$K = 5$	5	18.23 (0.02)	19.23 (0.05)	18.38 (0.06)	18.46 (0.06)	18.23 (0.07)
		$K = 10$	3	18.21 (0.01)	18.99 (0.04)	18.27 (0.04)	18.38 (0.06)	18.40 (0.05)
	$N = 40$	$K = 2$	20	26.88 (0.03)	29.55 (0.10)	26.58 (0.11)	28.91 (0.11)	28.73 (0.11)
		$K = 5$	8	27.43 (0.05)	30.85 (0.07)	28.37 (0.09)	29.21 (0.11)	29.55 (0.10)
		$K = 10$	4	27.54 (0.05)	30.53 (0.06)	27.47 (0.06)	28.17 (0.08)	28.75 (0.09)

Table 2: Benchmark Problem results for PSO, FEA and DFEA

		PSO	FEA	DFEA-Full
Benchmarks	Sphere	$3.8E+00$ $(1.7E-01)$	$1.7E-09$ $(1.5E-10)$	$1.5E-09$ $(1.4E-10)$
	Exponential	$-1.00E+00$ $(3.7E-05)$	$-1.0E+00$ $(1.7E-09)$	$-1.0E+00$ $(0.0E+00)$
	Schwefel	$4.4E+03$ $(9.6E+01)$	$1.8E+04$ $(1.2E+03)$	$2.5E+05$ $(6.3E+03)$
	Dixon-Price	$2.1E+01$ $(7.4E-01)$	$5.6E-01$ $(1.8E-01)$	$1.2E+00$ $(1.7E-01)$
	Ackley's	$2.2E+00$ $(4.5E-02)$	$1.9E-05$ $(8.4E-07)$	$2.3E-05$ $(1.9E-06)$
	Rosenbrock	$1.7E+02$ $(6.5E+00)$	$5.6E+00$ $(1.1E+00)$	$6.8E+00$ $(2.5E+00)$
	Griewank	$6.3E-01$ $(2.3E-02)$	$2.6E-03$ $(1.1E-03)$	$8.6E-02$ $(2.5E-02)$

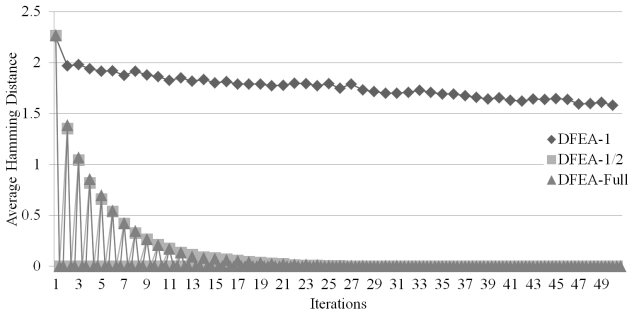

Figure 3: Average consensus between factors in DFEA on maximizing NK Landscapes $N = 25$ and $K = 10$.

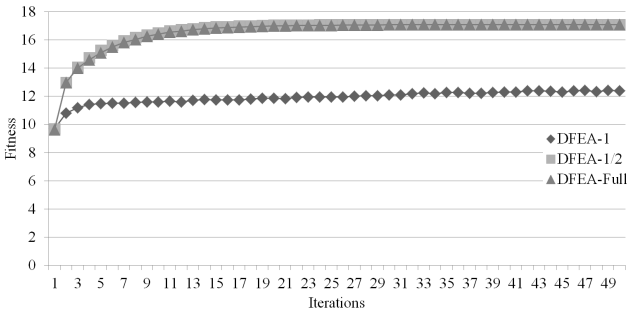

Figure 4: Fitness of DFEA on maximizing NK Landscapes $N = 25$ and $K = 10$.

petition steps in DFEA is small enough that only 1 sharing iteration is enough to reduce the difference between factors.

The results for the benchmark optimization problems are interesting. DFEA outperformed the PSO on all of the problems with results that are often several orders of magnitude better than PSO. For example, on Ackley's function, PSO achieved a mean minimum of 2.2E+00 whereas DFEA achieved a mean minimum of 2.3E-05. The one exception is the Schwefel 1.2 function where all three algorithms performed poorly. In general, DFEA results were the same or slightly worse than the FEA results.

As previously mentioned, DFEA-Full performed better on Sphere, Exponential, Dixon-Price, and Ackley's while DFEA-1 performed better on Rosenbrock and Griewank. It is not altogether clear why this would be the case because we would generally expect consensus to be more important on harder problems. Rosenbrock and Griewank are generally considered to be harder problems than Sphere or Exponential. The pattern for DFEA-1/2 is harder to summarize. It lies outside the DFEA-1 to DFEA-Full range on several problems (Sphere and Griewank), is sometimes closer to the winner (Exponential, Dixon-Price, Ackley's, Griewank) but is also sometimes closer to the loser (Rosenbrock).

7. CONCLUSION

We have been able to show that in problems with low epistasis, DFEA is able to perform well when using a reduced number of sharing iterations. However, for problems with high epistasis, DFEA performs worse with less sharing iterations. This drop in performance can be combated by performing more inter-factor iterations, but may negate the complexity reduction gained when the sharing iterations are reduced.

In our future work, we want to explore the relationship between performance and consensus of DFEA further. For example, we want to explore using the emergence of consensus between factors as a stopping criterion. We also want to see if minimum amount of consensus between factors needed

Table 3: Benchmark problem results with varying degrees of DFEA consensus

		DFEA-1	DFEA-1/2	DFEA-Full
Benchmarks	Sphere	$2.2E-09$ $(2.4E-10)$	$3.1E-09$ $(4.5E-10)$	$1.5E-09$ $(1.4E-10)$
	Exponential	$-4.7E-05$ $(3.6E-06)$	$-1.0E+00$ $(2.1E-09)$	$-1.0E+00$ $(0.0E+00)$
	Schwefel	$2.2E+05$ $(4.9E+03)$	$2.4E+05$ $(5.6E+03)$	$2.5E+05$ $(6.3E+03)$
	Dixon-Price	$9.6E+00$ $(1.8E+00)$	$1.2E+00$ $(1.9E-01)$	$1.2E+00$ $(1.7E-01)$
	Ackley's	$6.9E+00$ $(1.2E-01)$	$3.1E-05$ $(1.6E-06)$	$2.3E-05$ $(1.9E-06)$
	Rosenbrock	$4.8E+00$ $(2.7E-01)$	$7.5E+00$ $(2.6E+00)$	$6.8E+00$ $(2.5E+00)$
	Griewank	$4.7E-02$ $(5.4E-03)$	$4.1E-02$ $(1.7E-02)$	$8.6E-02$ $(2.5E-02)$

to guarantee DFEA converges at a similar rate to DFEA-Full. A related area of work involves investigating how some of the other FEA and DFEA parameters affect the algorithm performance. This includes looking at how to configure the number of times factors should optimize their values before competition and sharing are performed.

We also want to analyze the convergence properties of FEA and DFEA. Currently, it is not known whether FEA and DFEA are guaranteed of converging to a local optimum. To accomplish this, we plan on using Van den Bergh and Engelbrecht's convergence proof for Cooperative PSO and adapting it to FEA [17].

Finally, we plan to apply FEA and DFEA to a wider range of optimization problems including more benchmark optimization problems and other specific applications such as 3-SAT problems. This will help inform us as to what kind of problems FEA and DFEA are effective at solving.

Acknowledgments

This project was supported by an NSF Research Experience for Undergraduates grant, CNS-1156475, of which both Jenna Hoole and Benjamin Demeo were a part. We would like to thank the members of the Numerical Intelligence System Laboratory (NISL) at MSU, in particular, Dr. Nathan Fortier, Houston King, and Logan Perreault, for their advice and ideas as this research progressed.

8. REFERENCES

[1] N. Fortier. *Inference and Learning in Bayesian Networks Using Overlapping Swarm Intelligence*. PhD thesis, Montana State University, 2015.

[2] N. Fortier, J. Sheppard, and K. G. Pillai. Bayesian abductive inference using overlapping swarm intelligence. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS)*, pages 263–270, 2013.

[3] N. Fortier, J. Sheppard, and S. Strasser. Abductive inference in Bayesian networks using distributed overlapping swarm intelligence. *Soft Computing*, 19(4):981–1001, 2014.

[4] N. Fortier, J. Sheppard, and S. Strasser. Learning Bayesian classifiers using overlapping swarm intelligence. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS)*, 2015.

[5] N. Fortier, J. Sheppard, and S. Strasser. Parameter estimation in bayesian networks using overlapping swarm intelligence. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 9–16, 2015.

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

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

[8] K. James and E. Russell. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[9] M. Jamil and X. Yang. A literature survey of benchmark functions for global optimization problems. *CoRR*, abs/1308.4008, 2013.

[10] T. Jones. *Evolutionary algorithms, fitness landscapes and search*. PhD thesis, University of New Mexico, Department of Computer Science, 1995.

[11] S. A. Kauffman. *The origins of order: Self-organization and selection in evolution*. Oxford university press, 1993.

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

[13] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

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

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

[16] S. Strasser, J. Sheppard, and N. Fortier. Factored evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, Submitted 2015.

[17] F. Van den Bergh and A. P. Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):225–239, 2004.

[18] 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 (SIS)*, pages 141–149, 2007.