Convergence of Factored Evolutionary Algorithms

Shane Strasser
Gianforte School of Computing
357 Barnard Hall
Montana State University
Bozeman, MT 59717
shane.strasser@msu.montana.edu

John W. Sheppard
Gianforte School of Computing
357 Barnard Hall
Montana State University
Bozeman, MT 59717
john.sheppard@montana.edu

ABSTRACT

Factored Evolutionary Algorithms (FEA) have been found to be an effective way to optimize single objective functions by partitioning the variables in the function into overlapping subpopulations, or factors. While there exist several works empirically evaluating FEA, there exists very little literature exploring FEA's theoretical properties. In this paper, we prove that the final solution returned by FEA will be the results of converging to a single point. Additionally, we show how the convergence of FEA to a single point in the search space could be to a suboptimal point in space. However, we demonstrate empirically that when using specific factor architectures, the probability of converging to these suboptimal points in space approaches zero. Finally, where hybrid versions Cooperative Coevolutionary Algorithms have been proposed as a means to escape these suboptimal points, we show how FEA is able to outperform its hybrid version.

Keywords

Particle Swarm Optimization, Factored Evolutionary Algorithms, Convergence

1. INTRODUCTION

Many important problems involve function optimization, including bin packing, the traveling salesman problem, job shop scheduling, neural network training, and Bayesian network inference [26]. Often, stochastic search algorithms are used to solve such problems because the randomness used in the algorithms helps to escape local optima. One of the best-known families of stochastic search algorithms is the Evolutionary Algorithm (EA).

An example of an EA 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 [9]. During each iteration, candidate solutions undergo search operations such as crossover and mutation. Another popular EA that has been found to perform well on a variety of optimization problems is Differential Evolution (DE) [1]. The DE algorithm is similar to GA in that individuals undergo mutation, crossover, and selection, but differs in how those operations are performed.

Another population-based approach is called Particle Swarm Optimization (PSO) [12]. While 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 through the space.

While GA 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 schemata [15, 25]. Similarly, PSO can be prone to what is called "two steps forward and one step back," which happens when near optimal parts of an individual's current position may be thrown away if the rest of the individual's position causes the individual to have low fitness [30]. The consequences of hitchhiking and two steps forward and one step back is that the algorithm will often converge to suboptimal solutions [23].

One way to mitigate hitchhiking and "two steps forward and one step back" in GA, DE, and PSO is by generating subpopulations that optimize over subsets of variables [20]. Potter and De Jong presented one of the earliest such algorithms called the Cooperative Coevolutionary Genetic Algorithm (CCGA) that subdivided the problem by creating an individual GA that optimized over a single variable at a time. Other methods similar to CCGA have been presented, such as the Cooperative Particle Swarm Optimization (CPSO), which uses PSO as the underlying search algorithm and allows subpopulations to optimize over larger groups of variables [31]. However, one drawback to both of these methods is that they assume disjoint subpopulations.

Factored Evolutionary Algorithms (FEA) are a generalization of CCGA and CPSO that allow for subpopulations to overlap with one another [27]. 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 factors that when put together, represent full solutions to the problem. Additionally, FEA encourages the factors to overlap, which allows the factors to compete with one another for inclusion in a full solution Another unique property of FEA is that it allows for any population based algorithm to be used as the underlying optimization algorithm. [5]. Consequently, subpopulations or subswarms are

FOGA '17, January 12–15, 2017, Copenhagen, Denmark.

DOI: http://dx.doi.org/10.1145/3040718.3040727

associated with each factor, and the desired search algorithm is used to search the subspace covered by that factor.

Strasser *et al.* demonstrated that FEA is able to outperform other Cooperative Coevolutionary algorithms on a wide range of problems [27]. While FEA has been found to perform well, there is still little known about its convergence properties. For example, in previous work, the final solution returned by FEA always converged to a single point; however, it was unknown whether this would always be the case. Additionally, it was unknown if the final solution returned would be located at a global or local minimum. In this paper, we prove that FEA does converge to a single point, but that this point may not be a local or global minimum. We are also able to show that individual factors in FEA may become stuck at points in the search space called pseudominima. A pseudominimum is a point in the search space that appears as a local minimum in each subset of a configuration of subsets for the optimization problem, but is not a true local minimum for that problem as a whole.

Given these results, we demonstrate that, while FEA may theoretically not reach a local minimum, the likelihood of this occurring is often very small. To do so, we compare FEA with a hybrid version of CPSO that is guaranteed to reach a local minimum, called CPSO-H, which alternates between CPSO and PSO. By allowing a regular PSO to modify individuals in the subswarms in CPSO, CPSO-H is able to escape instances when subswarms become stuck at pseudominima. Additionally, we present a version of FEA called FEA-H, which alternates between FEA and the full EA. In doing so, we discover that while CPSO-H outperforms CPSO, the performance of FEA-H was usually equal or even worse than FEA.

There are several contributions made in this work. First, we prove under what conditions FEA will converge to a single solution. Second, we prove that FEA may converge to suboptimal solutions called pseudominima. Third, we define a hybrid FEA algorithm that combines FEA with a full population algorithm, called FEA-H. Finally, we present experimental results showing that while FEA may theoretically become stuck in pseudominima, the probability of this occurring is often very small.

The remainder of the paper is organized as follows. We first discuss relevant related work in Section 2 and then provide a detailed description of FEA in Section 3. In Section 4, we present our theoretical results that the full solution returned by FEA will converge to a single point if the factors also converge. Additionally, we show that this convergence may be to a pseudominimum. Next, in Section 5 we present a hybrid version of FEA, called FEA-H, that allows FEA to escape pseudominimum by alternating between FEA and a full single population EA. Finally, we compare FEA to Cooperative Coevolutionary algorithms as well as Cooperative Coevolutionary algorithms hybridized with single population algorithms in Section 6 and provide our conclusions and future work in Section 7.

2. RELATED WORK

Factored Evolutionary Algorithms (FEA) were first proposed by Strasser *et al.* as a framework for decomposing a single objective optimization problem into a set of subproblems that can be optimized by any evolutionary or swarm-based algorithm [27]. When put together, solutions to these subproblems (i.e., factors) represent full solutions to the problem. Additionally, FEA encourages the factors to overlap with one another, which allows the factors to compete for inclusion in the full solution [5, 27]. Strasser *et al.* demonstrated that FEA versions of Particle Swarm Optimization, Genetic Algorithms, Differential Evolution, and even simple Hill climbing outperformed single population and Cooperative Coevolutionary versions of the same algorithms. The authors also compared several different methods for deriving subpopulations for FEA and found that the architecture derived from a variable's Markov blanket outperformed the competing approaches. We make note that FEA has a similar name to the Fast Evolutionary Algorithm, which is a genetic algorithm that uses a fitness approximation routine to calculate the fitness of children based on the fitness of the parents, reducing the number of fitness evaluations [22].

One of the earliest versions of FEA was the Cooperative Coevolutionary Genetic Algorithm (CCGA) proposed by Potter and De Jong [20]. CCGA uses *subspecies* to represent non-overlapping subcomponents of a potential solution. Complete solutions are then built by assembling the current best subcomponents in a process called *collaboration*. The paper showed that, in most cases, CCGA significantly outperformed traditional GAs. Only in cases where the optimization function had high interdependence between the function variables (i.e., epistasis) did CCGA struggle because relationships between these variables were ignored.

More dynamic versions of CCGA have been proposed that allow for subpopulations to evolve over time [21]. 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 one another. However, this overlap is not guaranteed, and the algorithm does not have a means to resolve discrepancies between these overlapping subpopulations. The authors were able to demonstrate that their algorithm could evolve an effective number of subpopulations and was competitive with domain-specific algorithms on training cascade networks [21].

This idea of Cooperative Coevolutionary Evolutionary Algorithms (CCEA) was also applied by Van den Bergh and Engelbrecht when using PSO to train neural networks [30]. In their paper, the authors tested four fixed subswarm architectures of their own design. Comparing these four different architectures, the success of the algorithms was highly dependent on the architecture used, again due to the interdependence between the variables. By keeping variables with interdependencies together, the algorithm was more effective at exploring the search space [30]; however the subswarm architecture still did not allow for overlap.

Later, Van den Bergh and Engelbrecht extended their work by applying it to a wider range of optimization problems [31]. 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, the authors introduce a hybrid algorithm that alternates between CPSO and PSO. The result was an algo-

rithm that always outperformed PSO and was competitive with but more robust than CPSO.

Cooperative Coevolutionary (CC) algorithms have also been applied to Differential Evolution. Shi *et al.* proposed a simple and direct application of CCGA to DE, which they called CCDE [25]. Other adaptations have been more complex, such as those presented by Yang *et al.*, where the authors developed a weighted cooperative algorithm that used DE to optimize problems over 100 dimensions [34]. In particular, their weighting scheme allowed for evolving subpopulations. The authors' algorithm was found to outperform regular CC algorithms on most of the test functions explored [34].

Another variation of CCEA that used evolving subpopulations was proposed by Li and Yao [14]. Here, the subpopulation sizes were allowed to grow or to shrink when stagnation was detected, creating a wider range of variable groups. The authors 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 groups of random variables perform well, there should exist more intelligent ways of creating subpopulations.

CPSO was first adapted to allow explicitly for overlapping variables by Haberman and Sheppard in 2012 [8]. Here the authors developed an algorithm called Particle-based Routing with Overlapping Swarms for Energy Efficiency (PROSE), which used PSO as the underlying optimization algorithm. PROSE was presented as a method to develop energy-aware routing protocols for sensor networks that ensure reliable path selection while minimizing energy consumption during message transmission. Each subswarm in PROSE represented a sensor and all of the sensors' immediate neighbors in the network. PROSE was shown to be able to extend the life of the sensor networks and to perform significantly better than current energy-aware routing protocols.

Subsequently, PROSE was adapted by Ganesan Pillai and Sheppard to learn the weights of deep artificial neural networks [19]. In that work, the authors developed an algorithm called Overlapping Swarm Intelligence (OSI) where each swarm represents a unique path starting at an input node and ending at an output node. A common vector of weights, called the full global solution, 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. The authors 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.* called Distributed Overlapping Swarm Intelligence (DOSI) [7]. 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 was that a global solution was not used for fitness evaluation. The authors were able to show that DOSI's performance was close to that of OSI's on several different networks but there were instances when OSI outperformed DOSI. DOSI never outperformed OSI but was generally competitive.

OSI and DOSI have also been applied to the full and partial abductive inference problems in Bayesian networks, where the task is to find the most probable set of states for a set of nodes in the network given a set of observations [3, 5] The authors 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 the parameters or structure of Bayesian networks. For example, 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 [4]. Here each variable represents an attribute in the data, and for each variable, 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 significantly outperform competing structure learning approaches.

When learning Bayesian networks, latent or unobserved variables are often introduced, and Fortier *et al.* used OSI to learn the parameters of these latent variables [6]. A subswarm was created for each node with unknown parameters and all of the variables in that node's Markov blanket. The authors were able to show that OSI outperformed the competing approaches, including the traditionally-applied expectation-maximization algorithm, and that the amount of overlap between the subswarms can impact the performance of OSI.

There has been a limited number of papers published that have attempted to analyze theoretically the convergence of CCEAs, most of which have focused on collaboration in CCGA to evaluate individuals in a subpopulation [16]. For example, when evaluating an individual, which values should be pulled from other subpopulations to allow for calculating the fitness of the individual?

One of the earliest was by Wiegand *et al.*, who developed a framework to analyze convergence properties of CCEA using evolutionary game theory to model the interaction of two subpopulations [32]. The authors calculated the next generation by using a payoff matrix A and proved that when trajectories converge to a fixed point, the populations become homogeneous. Additionally, the authors showed that subpopulations may converge to points with "suboptimal fitness values." Finally, the authors used their framework to investigate empirically the effects of uniform crossover and bit flip mutation.

One of the drawbacks to this approach was that the authors assumed infinite population in each of the subpopulations and that an individual's fitness is given as the average fitness using all individuals from the other subpopulation during collaboration [17]. Panait *et al.* relax this requirement by modeling the fitness evaluation of an individual as the average fitness of N randomly chosen individuals from the other subpopulation. Additionally, the authors use evolutionary game theory as a way to visualize different convergence properties of CCEA [17].

Later work by Panait argued that the primary reason for poor performance in CCGAs is caused by poor selection of individuals during collaboration [16]. The authors go on to use a refined evolutionary game theory model and show that a CCGA will converge to globally optimal solution when the collaboration process is set properly. Additionally, Panait showed that a collaboration process that uses the best individuals from subpopulations outperforms a collaboration process that uses the average or worst individuals.

Other theoretical work involved with CCGAs has been to investigate its robustness. Wiegand and Potter first defined

a framework for characterizing robustness in evolutionary algorithms [33]. Using this framework, the authors were able to show that CCGAs exploit this robustness during search and demonstrate empirically how this is done [33].

As described elsewhere, Van den Bergh and Engelbrecht also presented work looking at the convergence of CCEAs [31], especially within the context of pseudominima. The authors present an extension to CPSO called CPSO-H, that avoids the problem of convergence to pseudominimum by iterating between CPSO and a full PSO algorithm, allowing the algorithm to escape pseudominima during the PSO portion of the algorithm. Because the CPSO-H iterates between CPSO and a regular PSO, the authors claim that any convergence results that are applicable to the full PSO are also applicable to CPSO-H [29].

Finally, we make note of some of the similarities between FEA and the Sequential Subspace Optimization (SESOP) algorithm [2]. SESOP is a gradient descent method that determines the next update based on a set of subspaces spanned by the current gradient and all previous steps. This is similar to FEA in that during each update, only a subset of variables may be updated. SESOP is different than FEA in that during each update, any subset of variables may be updated whereas in FEA, an individual is only able to update the same set of variables. Because SESOP is able to update any combination of variables, it is also less likely to become trapped in in pseudominima.

There are several different ways our theoretical work presented here differs from the previous work. The first is that prior work does not consider the overlap between factors (subpopulations). Second, we are currently not interested in the optimal way to evaluate an individual in a factor. Because there are multiple factors, the complexity to evaluate an individual with a large set of randomly selected values from other factors becomes intractable. FEA handles this problem by performing a local search during the competition step to generate a full global solution to allow for individuals to evaluate themselves. Finally, our work differs in that we enable any optimization algorithm to be used. All previous work assumes a particular algorithm, such as CPSO using PSO.

3. THE FACTORED EVOLUTIONARY ALGORITHM

In this section, we describe FEA. Note that a full specification of FEA can be found in [27]. There are three major subfunctions in FEA: update, competition and sharing. The update 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.

3.1 Factors

Given a function $f : \boldsymbol{D}^N \rightarrow \mathbb{R}$ with domain $\boldsymbol{D}^N$ to be optimized with parameters $\boldsymbol{X} = \langle X_1, X_2, \ldots, X_N \rangle$, let $\boldsymbol{S}_i$ be a subset of $\boldsymbol{X}$. Let K be the average size of all subsets $\boldsymbol{S}_i$. Without loss of generality, assume that all subsets $\boldsymbol{S}_i$ are the same size. A subpopulation or *factor* $\mathbf{P}_i$ can then be defined over the variables in $\boldsymbol{S}_i$ that are optimizing f.

Algorithm 1 FEA Compete

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

```
1: randVarPerm ← RandomPermutation(N)
2: for ranVarIndex = 1 to N do
3:     i ← randVarPerm[ranVarIndex]
4:     bestFit ← f(G)
5:     bestVal ← P_1[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:         P_j ← S_i[randPopPerm[ranPopIndex]]
10:        G[X_i] ← P_j[X_i]
11:        if f(G) is better than bestFit then
12:            bestVal ← P_j[X_i]
13:            bestFit ← f(G)
14:        end if
15:    end for
16:    G[X_i] ← bestVal
17: end for
18: return G
```

Note that f can still be optimized over the variables in $\boldsymbol{S}_i$ by holding variables $\boldsymbol{R}_i = \boldsymbol{X} \setminus \boldsymbol{S}_i$ constant. We refer to $\boldsymbol{R}_i$ as factor $\boldsymbol{S}_i$'s *remaining values*. An algorithm that uses a set of subpopulations to optimize a problem is called a multi-population algorithm. We denote the set of s factor in a multi-population algorithm as $\boldsymbol{\mathcal{S}} = \cup_{i=1}^{s} \boldsymbol{P}_i$.

When $s = 1$ and $\boldsymbol{S}_1 = \mathbf{X}$, 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 $s > 1$, $\boldsymbol{S}_i \subset \mathbf{X}$, and $\bigcup \boldsymbol{S}_i = \boldsymbol{X}$ for all populations, the algorithm becomes a multi-population algorithm. We define FEA to be an algorithm where the factors that are proper subsets of $\boldsymbol{X}$ and at least one factor overlaps with another factor.

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

3.2 Competition

The goal of competition in FEA is to find the factors with the state assignments that have the best fitness for each dimension. Here, we present the competition algorithm described by Strasser *et al.* [27] (Algorithm 1). FEA constructs a *full global solution* $\boldsymbol{G} = \langle X_1, X_2, \ldots, X_N \rangle$ that evaluates the optimized values from factors. For every $X_i \in \boldsymbol{X}$, the algorithm iterates over every factor containing X_i and finds the best value from those factors.

The algorithm first iterates over a random permutation of all of the variables in $\boldsymbol{X}$, 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 factors that are optimizing the variable X_i. Lines 10-14 then compare the individual values of variable

Algorithm 2 FEA Share

Input: Full global solution $\boldsymbol{G}$, factors $\boldsymbol{\mathcal{S}}$
Output: Updated factors $\boldsymbol{\mathcal{S}}$

1: **for all** $\boldsymbol{P}_i \in \boldsymbol{\mathcal{S}}$ **do**
2: **for all** $X_j \in \boldsymbol{R}_i$ **do**
3: $\boldsymbol{R}_i[X_j] \leftarrow \boldsymbol{G}[X_j]$
4: **end for**
5: $\boldsymbol{p}_w \leftarrow \boldsymbol{P}_i$.worst()
6: **for all** $X_j \in \boldsymbol{S}_i$ **do**
7: $\boldsymbol{p}_w[X_j] \leftarrow \boldsymbol{G}[X_j]$
8: **end for**
9: $\boldsymbol{p}_w.fitness \leftarrow f(\boldsymbol{p}_w \cup \boldsymbol{R}_i)$
10: **end for**
11: **return** $\boldsymbol{\mathcal{S}}$

X_i by substituting the factors' values into $\boldsymbol{G}$. In our implementation, the factor uses the best value found during the entire search process as its candidate value to be evaluated in lines 10-14. Note that this means changes are only made when the new fitness is strictly greater than the previous fitness. The values yielding the best fitness from the overlapping factors are saved and then inserted into $\boldsymbol{G}$. Once the algorithm has iterated over all variables in $\boldsymbol{X}$, the algorithm exits and returns $\boldsymbol{G}$.

Note that competition is not guaranteed to find the best combination of values from each factor, nor does it guarantee the combination of values is better than the previous $\boldsymbol{G}$. However, by iterating over random permutations of $\boldsymbol{X}$ and $\boldsymbol{\mathcal{S}}$, the algorithm is able to explore different combinations and is still able to find good combinations of values.

3.3 Sharing

The sharing step serves two purposes. First it allows overlapping factors to inject their current knowledge into other factors. Previous work by Fortier *et al.* discovered that this is one of the largest contributors to the FEA's performance [5]. Second, it sets each factor's $\boldsymbol{R}_i$ values to those in the full global solution $\boldsymbol{G}$ so that each factor $\boldsymbol{P}_i$ can evaluate its partial solution on f. The sharing algorithm is provided in Algorithm 2.

The share algorithm iterates over all the factors and updates each factor's remaining values by setting each variable $X_j \in \boldsymbol{R}_i$ to the value in $\boldsymbol{G}$ (lines 2-4). Next, the algorithm injects information from $\boldsymbol{G}$ into factor $\boldsymbol{P}_i$. To accomplish this, the algorithm finds the individual with the worst fitness in $\boldsymbol{P}_i$ (line 5). Then, the share algorithm sets the worst individual $\boldsymbol{p}_w$'s current position to the values in $\boldsymbol{G}$ (lines 6 - 8). Finally, the fitness for $\boldsymbol{p}_w$ is recalculated in line 9.

3.4 FEA

Now that the share and competition algorithms have been defined, we can give the full FEA (Algorithm 3). The algorithm works as follows. First, all of the subpopulations are initialized according to the optimization algorithm being used and the subpopulation architecture (line 1). The full global solution $\boldsymbol{G}$ is randomly initialized in line 2.

Next, the algorithm begins inter-factor optimization, which consists of three steps (lines 3-11). First, the algorithm iterates over each factor and optimizes the values using the corresponding search algorithm until some stopping criterion is met (line 6). The optimization of each individual factor is called the intra-population optimization step. Following

Algorithm 3 Factored Evolutionary Algorithm

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

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

intra-population optimization of all factors, competition occurs between factors in the Compete function on line 9. Finally, the Share function on line 10 shares the updated best states between the factors. These three inter-population optimization steps are repeated until the stopping criterion is met.

4. CONVERGENCE OF FEA

One open question concerning FEA is its convergence behavior. Previous work with FEA has found that the full global solution $\boldsymbol{G}$ always converged to a single solution; however, no one has analyzed under what conditions convergence will occur. Here, we present work showing that the full global solution $\boldsymbol{G}$ in FEA will converge to a single solution if the individual factors also converge. Additionally, we show that FEA may converge to suboptimal solutions that are not local minima.

4.1 Convergence to Single Solutions

First, we will prove under what conditions the full global solution $\boldsymbol{G}$ converges to a single solution. The following assumes the function being optimized is a minimization problem. We begin with some definitions.

DEFINITION 1. *Let Δ_i^t be the change in factor $\boldsymbol{S}_i$'s best position at time t where $\boldsymbol{S}_i^t = [s_{(i,1)}^t, s_{(i,2)}^t, \ldots, s_{(i,k)}^t]$ is the best position for factor $\boldsymbol{S}_i$ at time t, where each position $s_{(i,j)}$ corresponds to a parameter in the function $f : \boldsymbol{D}^N \rightarrow \mathbb{R}$ with domain $\boldsymbol{D}^N$ and parameters $\boldsymbol{X}$. The change in the position for a single factor $\boldsymbol{S}_i$ is calculated as*

$$\Delta_i^t = d(\boldsymbol{S}_i^{t-1}, \boldsymbol{S}_i^t) = \sqrt{\sum_{j=1}^{K} (s_{(i,j)}^{t-1} - s_{(i,j)}^t)^2}$$

where K is the size of the factor.

DEFINITION 2. *Let $df(\boldsymbol{S}_i)^t$ be the change in fitness in factor $\boldsymbol{S}_i$'s at time t, where*

$$df(\boldsymbol{S}_i)^t = f(\boldsymbol{S}_i^{t-1} \cup \boldsymbol{R}_i^{t-1}) - f(\boldsymbol{S}_i^t \cup \boldsymbol{R}_i^t).$$

where $\boldsymbol{R}_i^t$ are the set of values from $\mathbf{G}$.

Because $\boldsymbol{S}_i$ is only updated if the fitness is strictly less than the previous fitness, we know $df(\boldsymbol{S}_i)^t \geq 0$.

DEFINITION 3. *Let $\Delta_{\boldsymbol{G}}^t$ be the change in position for $\boldsymbol{G}$ where*

$$\Delta_{\boldsymbol{G}}^t = d(\boldsymbol{G}^{t-1}, \boldsymbol{G}^t) = \sqrt{\sum_{i=1}^{N} (g_i^{t-1} - g_i^t)^2}$$

and $\boldsymbol{G}^t = [g_1^t, g_2^t, \ldots, g_N^t]$.

DEFINITION 4. *Let $df(\boldsymbol{G})^t$ be the change in fitness of $\boldsymbol{G}$ at time t where*

$$df(\boldsymbol{G})^t = f(\boldsymbol{G}^{t-1}) - f(\boldsymbol{G}^t)$$

DEFINITION 5. *Let $\boldsymbol{D}_f$ represent the search space for the function f. Similarly, Let $\boldsymbol{D}_{Comp}^t$ be the search space for the competition algorithm at time t. Note that $\boldsymbol{D}_{Comp}^t$ is a discrete search space and is given by the set of best positions from all factors,*

$$\boldsymbol{D}_{Comp}^t = [D_1^t, D_2^t, \ldots, D_N^t]$$

where D_i^t is the set of values a variable X_i in $\boldsymbol{G}$ can assume at time t,

$$D_i^t = [s_{(1,i)}^t, s_{(2,i)}^t, \ldots, s_{(K,i)}^t]$$

and $s_{(1,i)}^t$ is the best position at time t for the first factor $\boldsymbol{S}_1$ that optimizes parameter X_i.

DEFINITION 6. *Let $\boldsymbol{C^t}$ be the set of all points in which $\boldsymbol{G}$ can assume at time t. The set $\boldsymbol{C^t}$ is determined by $\boldsymbol{D}_{Comp}^t$ and consists of*

$$\boldsymbol{C^t} = \{D_1^t \times D_2^t \times \cdots \times D_N^t\},$$

where

$$C_i^t = [s_{(a,1)}^t, s_{(b,2)}^t, \ldots, s_{(z,N)}^t]$$

and $s_{(a,1)}^t$ is a value for variable X_1 from a factor a. We denote the size of $\boldsymbol{C^t}$ as L.

DEFINITION 7. *A factor $\boldsymbol{S}_i$ converges when*

$$\lim_{t\to\infty} \Delta_i^t = 0.$$

Similarly, $\boldsymbol{G}$ is said to have converged when

$$\lim_{t\to\infty} \Delta_{\boldsymbol{G}}^t = 0.$$

THEOREM 4.1. *Assume that at time $t-1$, $\boldsymbol{G}$ is at a local optimum in $\boldsymbol{D}_{Comp}^{t-1}$. If $df(\boldsymbol{S}_i)^t = 0$ for all factors i, then $\Delta_{\boldsymbol{G}}^t = 0$ and $df(\boldsymbol{G})^t = 0$.*

PROOF. If $df(\boldsymbol{S}_i)^t = 0$, then no factors were updated and $\Delta_i^t = 0$ for all i. This indicates that the search space remain unchanged from $t-1$ to t and $\boldsymbol{C^{t-1}} = \boldsymbol{C^t}$. Since $\boldsymbol{G}$ was at a local optimum in $\boldsymbol{C^{t-1}}$, $\boldsymbol{G}$ is also at a local optimum in $\boldsymbol{C^t}$. Because there are no single changes that the competition algorithm can make to improve the fitness of $\boldsymbol{G}$, $\Delta_{\boldsymbol{G}}^t = 0$. Finally, because $\boldsymbol{G}$ remains unchanged, so does its fitness $df(\boldsymbol{G})^t = 0$. □

The above theorem shows that if the full global solution is locally optimal and the search space $\boldsymbol{D}_{Comp}^t$ does not change, then there are no changes that the competition algorithm can make to increase the fitness of $\boldsymbol{G}$. This leads us to the next theorem, which relates the convergence of factors to the convergence of $\boldsymbol{G}$.

THEOREM 4.2. *If $\boldsymbol{G}$ is at a local optimum in $\boldsymbol{D}_{Comp}^t$ and all factors have converged, then $\boldsymbol{G}$ has also converged.*

PROOF. By definition of convergence for a factor $\boldsymbol{S}_i$,

$$d(\boldsymbol{S}_i^{t-1}, \boldsymbol{S}_i^t) = 0$$
$$\boldsymbol{S}_i^{t-1} = \boldsymbol{S}_i^t.$$

Therefore,

$$f(\boldsymbol{S}_i^{t-1}) = f(\boldsymbol{S}_i^t)$$
$$f(\boldsymbol{S}_i^{t-1}) - f(\boldsymbol{S}_i^t) = 0$$
$$df(\boldsymbol{S}_i)^t = 0$$

for all factors i. By Theorem 4.1, $\boldsymbol{G}$ will not change and therefore has converged. □

The above theorem requires that all of the factors have already converged to guarantee the full global solution also converges. We relax this constraint in the next Theorem by only assuming that at some point in time the factors converge.

THEOREM 4.3. *If all the factors $\boldsymbol{S}_i$ in FEA converge at some point in time during FEA's optimization of f, then the full global solution $\boldsymbol{G}$ will also converge.*

PROOF. By definition of convergence for each factor $\boldsymbol{S}_i$,

$$\lim_{t\to\infty} \Delta_{\boldsymbol{S}_i}^t = 0.$$

Thus,

$$\lim_{t\to\infty} \sqrt{\sum_{j=1}^{N} ({s_{i,j}}^{t-1} - {s_{i,j}}^t)^2} = 0.$$

Because the distance function is positive semidefinite,

$$\lim_{t\to\infty} ({s_{i,j}}^{t-1} - {s_{i,j}}^t)^2 = 0$$

for all j in i. Let $\Delta_{\boldsymbol{C}}^t$ be the change in $\boldsymbol{A}$ at time t,

$$\Delta_{\boldsymbol{C}}^t = \frac{\sum_{i=1}^{L} d(C_i^{t-1}, C_i^t)}{L}$$

where C_i^t is the ith point in $\boldsymbol{C}$ at time t, and d is the Euclidean distance. Because $C_{i,j}^{t-1}$ is factor k's value for variable j, we know that the sequence of values will converge

$$\lim_{t\to\infty} ({S_{k,j}}^{t-1} - {S_{k,j}}^t)^2 = 0$$

therefore,

$$\lim_{t\to\infty} (C_{i,j}^{t-1} - C_{i,j}^t)^2 = 0.$$

for all j given i. Additionally,

$$\lim_{t\to\infty} d(C_i^{t-1}, C_i^t) = 0$$

for all factors i. Putting this together gives us

$$\lim_{t\to\infty} \frac{\sum_{i=1}^{M} d(\boldsymbol{C}_i^{t-1}, \boldsymbol{C}_i^t)}{M} = 0.$$

So,

$$\lim_{t\to\infty} \Delta_C^t = 0.$$

Recalling that this is a discrete search space with a finite number of points for the competition algorithm to explore,

this shows that the search space $\boldsymbol{D}^t_{Comp}$ also converges to a set of discrete points. Eventually, the competition algorithm, which is a local search algorithm (i.e., Hill Climbing), will either reach a local optimum or hit all points. Therefore, $\boldsymbol{G}$ will converge. □

The above theorem requires that all of the factors in FEA have converged. However, it is possible for the full global solution in FEA to converge even if not all of the factors have converged. Instead, we only require that $\boldsymbol{G}$ is a local optimum in all subsequent search spaces, which allows for some of the factors to not converge.

THEOREM 4.4. *If $\boldsymbol{G}$ is a local optimum in all spaces $\boldsymbol{D}^t_{Comp}$ $\forall t > t_0$, then $\boldsymbol{G}$ has converged.*

PROOF. Since $\boldsymbol{G}$ was at a local optimum in $\boldsymbol{C^t}$ for all future time past t_0, there are no single changes that the competition algorithm can make to improve the fitness of $\boldsymbol{G}$ during each FEA iteration. Therefore, $\Delta^t_{\boldsymbol{G}} = 0$, which by definition is the convergence of $\boldsymbol{G}$. □

4.2 Pseudominimum Convergence

The previous section showed that if the search space for the competition algorithm converges or if $\boldsymbol{G}$ is at a local optimum, then the full global solution $\boldsymbol{G}$ will converge to a single point. However, it is unknown to what kind of solutions the full global solution will converge. Additionally, while $\boldsymbol{G}$ may be located at a local optimum in $\boldsymbol{D}^t_{Comp}$, it may not be a local optimum in $\boldsymbol{D}^t_f$. Here, we prove that $\boldsymbol{G}$ may become stuck in suboptimal points in the search space known as pseudominima.

DEFINITION 8. *A* local minimum *is a point*

$$\boldsymbol{x}^* = (x_1, x_2, \ldots, x_N)$$

if there exists some $\epsilon > 0$ such that $f(\boldsymbol{x}^*) < f(\boldsymbol{x})$ for all points within a distance ϵ from $\boldsymbol{x}^*$.

DEFINITION 9. *Given a subspace $\mathcal{S}$ in $\mathbb{R}^K$ where $K < N$, a* pseudominimum *is a point*

$$\boldsymbol{x}_p = (x_1, x_2, \ldots, x_N)$$

such that $\mathbf{x}_p$ is a local minimum in the subspace $\mathcal{S}$ but is not a local minimum for f.

DEFINITION 10. *Given a subspace $\mathcal{S}$ in $\mathbb{R}^K$ where $K < N$, a* global pseudominimum *is a point*

$$\mathbf{x}_p = (x_1, x_2, \ldots, x_N)$$

such that $\mathbf{x}_p$ is a global minimum in the subspace $\mathcal{S}$ but is not a local or global minimum for f.

EXAMPLE 1. *An example of a global pseudominimum is the point $(0, 0)$ for the function*

$$f(\mathbf{X}) = g(h(\mathbf{X})) \tag{1}$$

where

$$g(\mathbf{X}) = X_1^2 + X_2^2 - (\tanh(10X_1) + 1)\,(\tanh(-10X_2) + 1) \exp\left(\frac{X_1 + X_2}{3}\right)$$

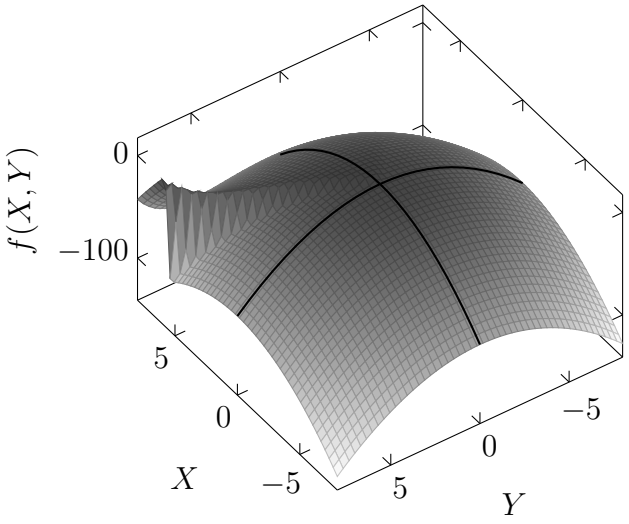

Figure 1: An inverse plot of Equation 1. Point $(0, 0)$ is an example of a global pseudominimum.

and $\mathbf{h}(\mathbf{X}, \theta)$ is a rotation operator defined as

$$\mathbf{h}(\mathbf{X}, \theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}.$$

To highlight the shape of the function, we plot the inverse in Figure 1. The point $(0, 0)$ is a global pseudominimum in the subspace defined by the y axis at $X = 0$ because moving in any direction in the y-axis, the function increases in value. However, the point is not a local minimum since the function can be decreased in the x and y axes simultaneously.

We also make note of a special set of pseudominima, called, maximal pseudominima, which are defined as follows.

DEFINITION 11. *Given a subspace $\mathcal{S}$ in $\mathbb{R}^K$ where $K < N$, a* maximal pseudominimum *is a point*

$$\mathbf{x}_p = (x_1, x_2, \ldots, x_N)$$

such that $\mathbf{x}_p$ is a local minimum in the subspace $\mathcal{S}$ but is not a local minimum for $\mathcal{S} \cup X_i$ for all variables in $\mathbb{R}^N$.

Definition 11 differs from Definition 9 in a key way. A pseudominimum in subspace $\mathcal{S}$ may also be a pseudominimum $\mathcal{S} \cup X_i$ where X_i is some dimension not in $\mathcal{S}$. The definition of a pseudominimum only requires that a point not be a local minimum for all dimensions. For example, a point may be a pseudominimum in two dimensions X_1 and X_2 for a function with 5 variables X_1, X_2, X_3, X_4, X_5. But it may also be a pseudominimum in the dimensions X_1, X_2, and X_3. If the point is not a pseudominimum in the subspaces X_1, X_2, X_3, X_4 and X_1, X_2, X_3, X_5, then it is a maximal pseudominimum.

We note that pseudominima are similar to saddle points. While saddle points are pseudominima, not all pseudominima are saddle points. For example, $(0, 0)$ in Figure 1 is a pseudominimum but is not a saddle point.

With these definitions, we can now give the following theorems. First, we show that there exist global pseudominima that will trap FEA. The second theorem generalizes this existence theorem by showing under what conditions FEA will become trapped by global pseudominima.

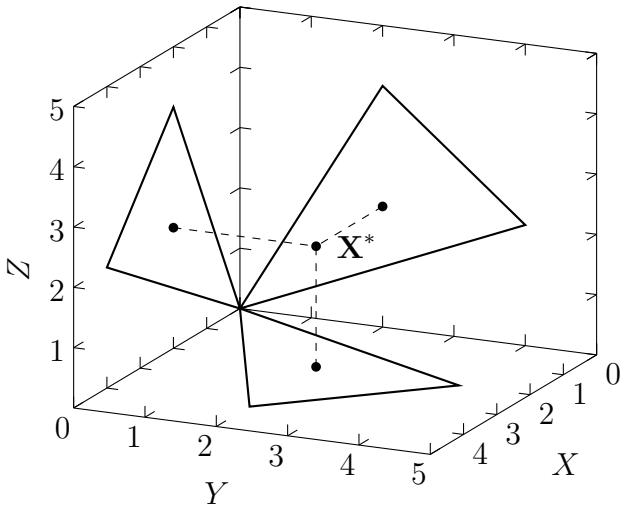

Figure 2: Three-dimensional function with a pseudominimum when projected into primary planes.

THEOREM 4.5. *There exist global pseudominima that will trap FEA.*

PROOF. We prove this by example. Assume we are given a function $f : \boldsymbol{D}^N \rightarrow \mathbb{R}$ with domain $\boldsymbol{D}^N$, that the output of the function is greater than 0, and the function is strictly increasing when moving in any direction from the origin. Additionally, there exists a simplex with a point at the origin where the rest of the points that define the simplex are strictly greater than 0. At the the tip of simplex, the output of f is equal to the origin, and within the simplex, the fitness decreases from all of the points that define the simplex to a local minimum $\boldsymbol{X}^* = (X_1^*, X_2^*, \ldots, X_N^*)$.

An example function with 3 dimensions is shown in Figure 2 with the sides of the triangle region projected onto the primary planes. In this example, the function for three variables X, Y and Z, is equal to $f(\boldsymbol{X}) = ||\boldsymbol{X}||$. The only exception is in a triangle shaped region where $f(\boldsymbol{X}) < 0$. Let B be the simplex region. We represent the function as

$$f(\boldsymbol{X}) = \begin{cases} g(\boldsymbol{X}) & \text{if } B \text{ contains } \boldsymbol{X} \\ \sqrt{X^2 + Y^2 + Z^2} & \text{else} \end{cases}$$

where

$$g(\boldsymbol{X}) = (X - X^*)^2 + (Y - Y^*)^2 + (Z - Z^*)^2 - C.$$

C is a constant value to ensure that all values of the function g within the simplex are negative.

Suppose an FEA is applied with factors $S_1,\ S_2,\ \ldots S_M$ where each factor S_i optimizes over a pair of variables X_j, X_k. Note that the origin $(0, 0 \ldots, 0)$ is a global pseudominimum defined by the FEA's factors since moving in only two directions causes f to increase in value. However, the origin is not a local minimum since f decreases in value by moving in all N directions simultaneously. thereby moving into the simplex with negative values.

Suppose the FEA has a full global solution G at the origin $(0, 0, \ldots, 0)$. If during factor $\boldsymbol{S}_1$'s solve step it evaluates the point X_1^*, X_2^*, the fitness will be some value L. By definition of f, the output is strictly increasing when moving away from the origin, and therefore, L is greater than 0. This is because FEA uses the full global solution $\boldsymbol{G}$ to evaluate the values X_1^*, X_2^*.

In the example shown in Figure 2, this would equate to three factors: $\boldsymbol{S}_1, \boldsymbol{S}_2$, and $\boldsymbol{S}_3$, each of which optimizes over variables XY, YZ and XZ, respectively. Additionally, $\boldsymbol{G} = (0, 0, 0)$. Factor $\boldsymbol{S}_1$ would evaluate the point X^*, Y^* as

$$f(X^*, Y^*, 0) = \sqrt{(X^*)^2 + (Y^*)^2}.$$

Note that this point is not within the simplex because all points that define the simplex are strictly greater than 0.

Because the fitness of X^*, Y^* is greater than 0 and the current best fitness of the factor, the factor discards this point and X_1^*, X_2^* will not be used in the competition and sharing steps. Consequently, the full global solution is unable to move from the origin to $(X^*, Y^*, 0, \ldots, 0)$. This same phenomenon will also occur for all factors; therefore, FEA will be stuck at the global pseudominimum. □

THEOREM 4.6. *Given an FEA, let $\boldsymbol{\mathcal{S}}$ be a set of subspaces $\mathbb{R}^K$ as defined by the set of factors in the FEA. If FEA's full global solution reaches a point $\boldsymbol{p}$ that is a global pseudominimum in all subspaces $\mathcal{S}_i \in \boldsymbol{\mathcal{S}}$, then FEA will be unable to escape $\boldsymbol{p}$.*

PROOF. Assume we are given a function $f : \boldsymbol{D}^N \rightarrow \mathbb{R}$ with domain $\boldsymbol{D}^N$ and that the FEA has a set of factors $S_1,\ S_2,\ \ldots S_M$. Also, assume that the full global solution $\boldsymbol{G} = (X_1^*, X_2^*, \ldots X_N^*)$ is at a point $\boldsymbol{p}$ that is a global pseudominimum. By definition of a global pseudominimum, each factor $\boldsymbol{S}_i$ will be unable to move to a better position because all other points in the subspace that factor $\boldsymbol{S}_i$ is searching over will have fitness greater than the current fitness of $\boldsymbol{S}_i$ at $\boldsymbol{p}$. Since each factor will be unable to locate a position with better fitness than its current position, the factor will not use other values different than those in $\boldsymbol{G}$ during the competition and sharing steps of FEA. Because no other values are used during competition, the full global solution is unable to move from the global pseudominimum; therefore, FEA will be unable to escape from the global pseudominimum. □

While this shows that FEA may become stuck, the factors must be optimizing over the variables that induce the pseudominimum. For example, if a factor is optimizing over all 3 variables in Figure 2, then the factor using hill climbing as the algorithm will not be trapped by the point (0,0,0). This suggests that if it is known where a pseudominimum occurs, then there should exist a factor that optimizes over a superset of variables that induce the pseudominimum. Note also that this only works if the pseudominimum is maximal.

Another consequence of this result is that it may provide another explanation of the results presented by Strasser *et al.* as to why certain factor architectures outperform others [27]. In that work, the authors hypothesized that the best factor architecture performed better than the other architecture because the better performing architecture minimized hitchhiking. However, the performance of the better factor architecture may also be due to fact that the factors have less chance of becoming trapped in pseudominima.

5. HYBRID FEA

In the previous section, we showed that the full global solution $\mathbf{G}$ in FEA will converge to a single solution. However, similar to CPSO, FEA may converge to suboptimal

Algorithm 4 Hybrid Factored Evolutionary Algorithm

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

```
1: S ← initializeFactors(f, X, A)
2: G ← initializeFullGlobal(S)
3: repeat
4:     for all S_i ∈ S do
5:         repeat
6:             S_i.updateIndividuals()
7:         until Termination criterion is met
8:     end for
9:     G ← Compete(f, S)
10:    Share(G, S)
11:    Full.seed(G)
12:    repeat
13:        Full.updateIndividuals()
14:    until Termination criterion is met
15:    if Full.bestFitness() is better than f(G) then
16:        G ← Full.bestSolution()
17:    end if
18: until Termination criterion is met
19: return G
```

solutions called pseudominima that regular EA are able to escape. As described earlier, van den Bergh and Engelbrecht proposed an extension to the CPSO algorithm called Hybrid Cooperative Particle Swarm Optimization (CPSO-H). CPSO-H combined the benefits of CPSO and a regular PSO by performing a number of iterations using CPSO followed by running a regular PSO. By performing several iterations with a full PSO, CPSO-H is able to escape pseudominima because an individual is able to modify all variables in one round of updates. Here, we present a similar extension to FEA called FEA-H (Algorithm 4).

FEA-H works as follows. First, it performs the same set of operations as FEA—Update, Compete, and Share—in lines 1–10. Next, FEA-H performs a set of updates to the full EA population, which is denoted as *Full*. First, the position of the individual with the worst fitness from *Full* is set to the full global solution **G** (line 11). In line 14, the full population updates its individuals until some stopping criteria is satisfied. Finally, the fitness of the best solution in *Full* is compared with **G**. If the fitness of the best individual from the full population is better than the full global solution, then the algorithm sets **G** to solution with the better fitness (line 16). However, if the fitness is not better, then no changes are made to **G**.

By running a set of iterations with a full EA, FEA-H is should be able to escape pseudominima. This is because when *Full* updates its individuals, each individual has the opportunity to change every variable simultaneously. For example, in Figure 1, the full population is able to move individuals from the origin to the optimal solution by following the incline on the ridge. However, if the factor architecture subsumes the pseudominima in the fitness landscape, then FEA will not become stuck at suboptimal solutions, and FEA-H will provide little to no benefit over FEA.

6. EMPIRICAL ANALYSIS

As shown in Section 4, FEA is still susceptible to pseudominima. However, FEA only becomes stuck at these points if the factors are optimizing over a subset of variables in $\mathbb{R}^k$. We hypothesize that for certain factor architectures in FEA, the probability of pseudominima becomes low. To test this hypothesis, we compare versions of CPSO and FEA with the hybrid version of CPSO-H presented by Van de Bergh and Englebrecht [31]. In addition, we compare all algorithms with FEA-H presented in Section 5. If FEA does become stuck in pseudominima, then one would expect FEA-H to outperform FEA because the full population in FEA-H allows the algorithm to escape pseudominima. However, if FEA does not become stuck in pseudominima, FEA-H will provide little to no benefit over FEA.

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 algorithms applied to epistatic functions. We included abductive inference in Bayesian Networks because they are a practical combinatorial optimization problem.

Because the focus in this set of experiments is to analyze the convergence of FEA, we restrict our analysis to using PSO as the underlying search algorithm. Previous work by Strasser *et al.* has already demonstrated the performance of FEA versions with Genetic Algorithms, Differential Evolution, Particle Swarm Optimization, and simple Hill Climbing over single population and Cooperative Coevolutionary versions. On the NK landscapes and abductive inference, we used a modified version of PSO since both problems are functions with a discrete input. To handle these problems, we used the Integer and Categorical Particle Swarm Optimization (ICPSO) algorithm proposed by Strasser *et al* as the underlying search algorithm [28].

6.1 Integer and Categorical PSO

The Integer and Categorical Particle Swarm Optimization (ICPSO) algorithm is a new PSO algorithm developed by Strasser *et al.* that has been shown to outperform other discrete PSO algorithms [28]. The ICPSO algorithm incorporates ideas from Estimation of Distribution Algorithms (EDAs) in that particles in the PSO represent probability distributions rather than solution values, and the PSO update modifies the probability distributions. This differs from other PSO variants, where a particle's position is often a direct representation of the solution values.

In ICPSO, a particle p's position is represented as

$$\boldsymbol{X}_p = [\mathcal{D}_{p,1}, \mathcal{D}_{p,2}, \ldots, \mathcal{D}_{p,n}]$$

where each $\boldsymbol{D}_{p,i}$ denotes the probability distribution for variable X_i. In other words, each entry in the particle's position vector is itself comprised of a set of distributions:

$$\boldsymbol{D}_{p,i} = [d^a_{p,i}, d^b_{p,i}, \ldots, d^k_{p,i}],$$

where $d^j_{p,i}$ corresponds to the probability that variable X_i takes on value j for particle p.

A particle's velocity is a vector of n vectors ϕ, one for each variable in the solution, that adjust the particle's probability distributions.

$$\boldsymbol{V}_p = [\phi_{p,1}, \phi_{p,2}, \ldots, \phi_{p,n}]$$
$$\phi_{p,i} = [\psi^a_{p,i}, \psi^b_{p,i}, \ldots, \psi^k_{p,i}].$$

where $\psi^j_{p,i}$ is particle p's velocity for variable i in state j. The velocity and position update equations are identical to those of traditional PSO and applied directly to the continuous values in the distribution.

$$\begin{aligned}\boldsymbol{V_p} &= \omega \boldsymbol{V_p} + U(0,\phi_1) \otimes (\boldsymbol{pBest} - \boldsymbol{X_p}) \\ &\quad + U(0,\phi_2) \otimes (\boldsymbol{gBest} - \boldsymbol{X_p}) \\ \boldsymbol{X_p} &= \boldsymbol{X_p} + \boldsymbol{V_p}\end{aligned}$$

The difference operator is defined as a component-wise difference between the two position vectors, i.e. for each variable X_i and value $j \in Vals(X_i)$, $d^j_{(\mathbf{pBest_p}-\mathbf{P_p}),i} = d^j_{pB,i} - d^j_{p,i}$. Here, $d^j_{p_B}$ is the personal best position's probability that variable X_i takes value j. The global best equation is identical except $\boldsymbol{pBest_p}$ is replaced with $\boldsymbol{gBest}$ and $d^j_{pB,i}$ with $d^j_{gB,i}$. The addition of the velocity vector to the position vector is similarly component-wise over each value in the distribution. For each probability for variable X_i and possible value j, the addition is $d^j_{p,i} + \psi^j_{p,i}$.

After the velocity and position update, an extra check is performed to ensure that probabilities fall within $[0, 1]$. Additionally, the distribution is normalized to ensure that its values sum to 1.

To evaluate a particle p, its distributions are sampled to create a candidate solution

$$\boldsymbol{S}_p = [s_{p,1}, s_{p,2}, \ldots, s_{p,n}]$$

where $s_{p,j}$ denotes the state of variable X_j.

The fitness function is used to evaluate the sample's fitness, which then is used to evaluate the distribution. When a particle produces a sample that beats the global or local best, both the distributions from that particle's position, $\boldsymbol{P}_p$, and the sample itself, $\boldsymbol{S}_p$, are used to update the best values. Mathematically, for all states $j \in Vals(X_i)$ the global best's probability is updated as

$$d^j_{gB,i} = \begin{cases} \epsilon \times d^j_{p,i} & \text{if } j \neq s_{p,i} \\ d^j_{p,i} + \sum\limits_{\substack{k \in Vals(X_i) \\ \wedge k \neq j}} (1-\epsilon) \times d^k_{p,i} & \text{if } j = s_{p,i} \end{cases}$$

where ϵ, the *scaling factor*, is a user-set parameter that determines the magnitude of the shift in the distribution restricted to $[0, 1)$. This increases the likelihood of the distribution producing samples similar to the best sample, while inherently maintaining a valid probability distribution. The procedure for setting the local best is directly analogous. The global best sample is returned as the solution at the end of optimization.

6.2 Test Problems

6.2.1 NK landscapes

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 [11]. It is defined by 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. In our work, we used 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. We generated NK landscapes with parameters $N = 25, 40, 50$, and $K = 5$. For each set of parameters, we created 30 random landscapes and ran each algorithm 30 times.

6.2.2 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 [13, 18]. 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 local probability distribution conditioned on the node's parents.

A common type of query for Bayesian networks is called abductive inference, which finds the most probable state assignment to a set of unobserved variables given a set of observed variables (evidence). To evaluate the fitness of a state assignment in abductive inference, we used the log likelihood ℓ, which is calculated

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

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. However, in all our experiments, we used an empty evidence set to minimize the number of parameters in the experiment. For test networks, we used the Alarm, Andres, Child, Hailfinder, Hepar2, Insurance, and Win95pts Bayesian networks from the Bayesian Network Repository [24].

6.2.3 Benchmark Optimization Problems

For the benchmark functions, we chose the following: Ackley's, Dixon Price, Exponential, Griewank, Rastrigin, Rosenbrock, Schwefel 1.2, and Sphere [10]. All of the problems are minimization problems with optimal finesses of 0.0 except for Exponential, whose optimal fitness is -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}. All functions were tested with 30 dimensions.

6.3 Setup

For the FEA algorithm on the NK landscapes and Bayesian networks, we used the Markov blanket factor architecture proposed by Fortier *et al.* since this was shown by Strasser *et al.* to outperform all other architectures on Bayesian networks and NK landscapes [5, 27]. On the Benchmark functions, we used the Simple Centered (SC) architecture of size two proposed by Strasser *et al.* since SC architecture had the most consistent performance over all functions [27]. The SC architecture creates a factor for each neighboring pairs

of variables in an ordered list of variables corresponding to the function definition.

For the CPSO algorithms, we had each subswarm optimize over two variables in the problem. This subswarm size was found to have the most consistent performance during the tuning of the algorithms. Furthermore, even when other subswarm sizes performed better than those used here, the differences were not significant.

Each algorithm was given a total of 400 individuals to divide between their subswarms. For the hybrid algorithms CPSO-H and FEA-H, an additional 10 individuals were used for the full algorithm step of the algorithms. These values were found to perform well for all algorithms during tuning. On the NK landscapes and abductive inference, both versions of CPSO and FEA used ICPSO as the underlying search 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. In ICPSO, the scaling value ϵ was set to 0.75. These values were found to perform well for all algorithms on all problems during tuning of the algorithms.

6.4 Results

Table 1 shows the results comparing CPSO, CPSO-H, FEA, and FEA-H on maximizing NK landscapes. Results from abductive inference on Bayesian networks are shown in Table 2 Note that both these problems are maximization. Results comparing CPSO, CPSO-H, FEA, and FEA-H on minimizing the benchmark functions are in Table 3. All results are expressed as means over 30 trials with standard errors in parentheses. Bold values indicate a significant difference between the regular CPSO or FEA algorithms with the hybrid versions using Mann-Whitney U test with $\alpha = 0.05$. Note that we did not perform any statistical testing between CPSO and FEA because it has already been shown by Strasser *et al.* that FEA outperforms CPSO.

As we can see in the NK landscape results, CPSO-H outperformed CPSO on two out of the three landscapes, but was only significantly better on $N = 25, K = 5$. Only on the larger problems ($N = 50$) did CPSO-H tie with CPSO-H. However, when looking at the FEA, the hybrid version of FEA was always outperformed by regular FEA, so hybridization provided no benefit and appears to have hurt performance.

The Bayesian network results show a similar trend in comparing regular and hybrid versions of CPSO and FEA. On the Andes, Hailfinder, Insurance, and Win95pts networks, CPSO-H outperformed CPSO significantly. Additionally, it was not outperformed by CPSO on the Alarm and Child networks. Only on the Hepar2 networks did CPSO significantly outperform CPSO-H. FEA outperformed FEA-H significantly on the Andes, Hailfinder, and Win95pts networks. Furthermore, it was only outperformed by FEA-H on the Child and Hepar2 networks. Note that in both of these cases, there was no significant difference between FEA and FEA-H.

On the benchmark functions, CPSO-H outperforms CPSO significantly on Griewank and Schwefel whereas, CPSO performs significantly better than CPSO-H on the Ackleys, Dixon Price, Exponential, Rastrigin and Sphere functions. FEA outperformed FEA-H by a significant margin on all functions except the Griewank function, where FEA-H outperformed FEA. We also note that the trends in the differences between CPSO and CPSO-H are similar to that of FEA and FEA-H. Only on the Schwefel function was there a significant difference in the trends of CPSO and CPSO-H with that of FEA and FEA-H.

6.5 Analysis

From the NK landscape, Bayesian network, and benchmark results, we see that the full PSO steps used by FEA-H provided a performance gain only a few times. In particular, FEA-H only significantly outperformed FEA on the Griewank function. Additionally, in many of the problems, the full population steps in FEA-H hurt the performance.

While CPSO-H significantly outperformed CPSO on the majority of the NK landscapes and Bayesian networks, CPSO performed significantly better than CPSO-H on the majority of the benchmark functions. There are two possible reasons for this result. One is that in the majority of the functions, there are fewer pseudominima that trap CPSO; therefore, CPO-H provides fewer benefits than regular CPSO. The other possible reason is that the creation of the subswarms for CPSO leads to the ability to avoid the pseudominima in the search space on the majority of the functions.

Another result we would like to make note of is the similarities between CPSO and FEA on the benchmark functions. We believe that the similarity of these results is due to the subswarm (factor) size for both CPSO and FEA being set to two thus not adequately capturing all of the variable interactions. Even with these similarities, on the Rosenbrock and Schwefel functions, CPSO did not outperform CPSO-H whereas FEA outperformed FEA-H on both these functions. Specifically, with Rosenbrock and Schwefel functions the overlap in FEA appears to allow the subpopulations to capture the majority of the of the variable interactions that CPSO is unable to capture. Again, this is because CPSO subpoulations optimize only disjoint sets of variables.

For the NK landscape and Bayesian networks, we believe that the performance of FEA over FEA-H is because the factors in FEA are less susceptible to pseudominima than the subswarms in CPSO. In the benchmark functions, CPSO was able to escape the majority of the pseudominima. But on the NK landscapes and Bayesian networks, CPSO had a higher liklihood of becoming trapped in pseudominima, which explains why CPSO-H often outperformed CPSO. Meanwhile, the factors in FEA are less prone to get stuck in pseudominima because they are optimizing over larger groups of subspaces that induce the pseudominima; therefore, the full PSO steps are not needed.

We explored this hypothesis further by running an experiment where, during FEA, we checked to see if the factors' best solutions were at a pseudominimum after the compete step. Because the Bayesian networks and NK landscapes are discrete problems, we are able to look at all neighboring states of a factor. For a given factor $\boldsymbol{S}_i$, if there does not exist a neighboring state with better fitness, then $\boldsymbol{S}_i$ could be at a pseudominimum. However, this point could also be a true local minimum. To see if $\boldsymbol{S}_i$ is in fact at a pseudominimum, we check to see if any neighboring points of $\boldsymbol{S}_i \cup \boldsymbol{R}_i$ have better fitness. If there does exist a neighboring point of $\boldsymbol{S}_i \cup \boldsymbol{R}_i$ with better fitness, then $\boldsymbol{S}_i$ is a true pseudominimum. However, if there are no neighboring points of $\boldsymbol{S}_i \cup \boldsymbol{R}_i$ with better fitness, then $\boldsymbol{S}_i$ is at a local minimum and not a pseudominimum. If the factor is not at a pseudominima or local minima, it is ignored in the calculation. This is because

Table 1: Results from comparing regular and hybrid versions of CPSO and FEA on NK landscapes.

	CPSO	CPSO-H	FEA	FEA-H
N=25, K = 5	1.78E+01(6.06E−02)	**1.83E+01(3.80E−02)**	**1.91E+01(3.15E−02)**	1.89E+01(3.19E−02)
N=40, K = 5	2.81E+01(1.09E−01)	2.86E+01(7.00E−02)	**3.05E+01(4.72E−02)**	3.01E+01(5.08E−02)
N=50, K = 5	**3.47E+01(1.31E−01)**	3.47E+01(6.74E−02)	**3.81E+01(3.87E−02)**	3.65E+01(5.11E−02)

Table 2: Results from comparing regular and hybrid versions of CPSO and FEA on Bayesian networks.

	CPSO	CPSO-H	FEA	FEA-H
Alarm	−1.99E+01(2.01E+00)	−1.59E+01(1.49E+00)	−9.12E+00(5.44E−01)	−9.87E+00(6.45E−01)
Andes	−2.01E+02(8.72E+01)	**−1.72E+02(7.12E+00)**	**−7.37E+01(8.34E−01)**	−8.80E+01(1.30E+00)
Child	−9.54E+00(5.08E−01)	−9.06E+00(4.48E−01)	−6.61E+00(3.03E−01)	−6.57E+00(2.98E−01)
Hailfinder	−7.39E+02(1.60E+02)	**−2.28E+02(6.92E+01)**	**−3.43E+01(3.61E−01)**	−3.57E+01(3.18E−01)
Hepar2	**−1.86E+01(4.48E−01)**	−2.14E+01(7.96E−01)	−1.81E+01(4.35E−01)	−1.76E+01(3.04E−01)
Insurance	−3.26E+02(8.56E+01)	**−1.57E+01(9.07E−01)**	−9.65E+00(4.13E−01)	−1.04E+01(2.91E−01)
Win95pts	−2.65E+02(1.10E+02)	**−1.17E+02(3.46E+01)**	**−1.82E+01(5.90E−01)**	−3.26E+01(1.16E+00)

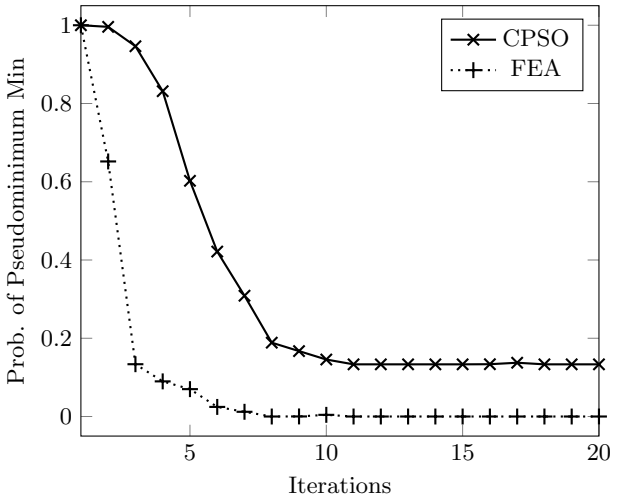

Figure 3: Probability of pseudominimum for NK landscape N = 25, K = 5.

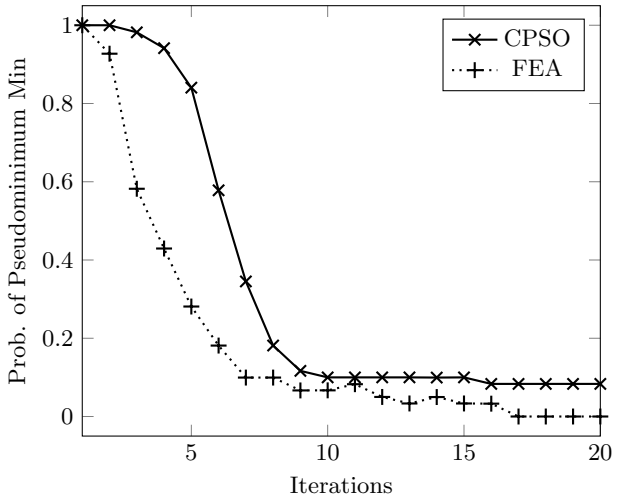

Figure 4: Probability of pseudominimum for Insurance network

a factor not at a local or pseudominimum suggests that the factor is still in the process of moving towards a better area in the search space.

We ran these experiments on the NK landscape with $N = 25$, $K = 5$, and on the Insurance network. Results for both CPSO and FEA are shown in Figures 3 and 4 respectively. In both these figures, the X-axis represents the number of iterations of the evolutionary algorithm. The Y-axis gives the probability of a pseudominimum at each iteration and is calculated as

$$\frac{\#pm}{\#pm + \#lm}$$

where $\#pm$ is the number of factors at a pseudominimum and $\#lm$ is the number of factors at a local minimum. A value of 1 means that all subswarms and factors are located at pseudominima while 0 indicates that all factors are located at local minima. Note that there is a possibility that the probability of a pseudominimum may become undefined if none of the factors are at a local minima or pseudominima. However, those instances were never encountered in these experiments.

As we can see in the two figures, both CPSO and FEA begin with a high probability of being located at pseudominima. This is likely due to the fact that the subswarms have just begun to locate good areas in the search space and are still moving towards those areas. However, we can see that the probability of factors being located at pseudominima in FEA decreases much faster than CPSO. Additionally, as the number of iterations increase, the probability for FEA becomes closer to zero. While the probability of CPSO does decrease over time, there is approximately a 20% probability of a subswarm being located at a pseudominimum. Finally, we note that the probability of pseudominima never increases because the definition of pseudominimum excludes a local minimum. Once a factor reaches a local minimum, it becomes more difficult for the factor to escape the local minimum, thus reducing the likelihood of a factor moving from a local minimum to a pseudominimum.

These results highlight why CPSO-H sees greater performance gains over CPSO than FEA-H does over FEA. On the NK landscapes and Bayesian networks, CPSO becomes trapped in pseudominima. CPSO-H is able to escape these

Table 3: Results from comparing regular and hybrid versions of CPSO and FEA on benchmark functions.

	CPSO	CPSO-H	FEA	FEA-H
Ackleys	**6.47E−06(3.86E−07)**	5.61E−05(6.24E−06)	**6.12E−06(7.39E−07)**	9.90E−05(1.20E−05)
Dixon Price	**2.22E−02(2.22E−02)**	2.17E−01(1.52E−01)	**4.44E−02(3.09E−02)**	1.56E−01(5.23E−02)
Exponential	**−1.00E+00(2.95E−10)**	−1.00E+00(1.02E−08)	**−1.00E+00(1.47E−10)**	−1.00E+00(7.34E−08)
Griewank	3.17E−02(1.95E−02)	**3.11E−03(1.50E−03)**	4.77E−02(8.32E−03)	**5.33E−03(1.99E−03)**
Rastrigin	**4.34E+00(3.83E−01)**	5.27E+00(3.76E−01)	**3.38E+00(2.96E−01)**	4.51E+00(4.49E−01)
Rosenbrock	2.81E+00(5.77E−01)	2.28E+01(1.40E+01)	**9.29E+00(3.07E+00)**	4.80E+01(7.22E+00)
Schwefel	2.86E+05(5.31E+04)	**1.42E+03(5.08E+02)**	**6.06E+02(5.63E+01)**	1.00E+03(6.96E+01)
Sphere	**2.15E−08(6.64E−09)**	7.48E−07(1.68E−07)	**8.97E−09(1.20E−09)**	3.21E−06(1.15E−06)

pseudominima and continue searching towards better solutions. FEA, on the other hand, has a smaller probability of becoming stuck in a pseudominima; therefore, the full swarm steps in FEA-H provide less benefit because the algorithm is already able to move towards good locations in the search space.

While the results suggest that FEA is less prone to pseudominima, the benchmark results suggest that the architecture is not the best for all functions. This is demonstrated by FEA-H significantly outperforming FEA on the Griewank function. We believe that main cause is that the factor architecture used for the Griewank function is suboptimal, and given a better factor architecture, we may see FEA outperform FEA-H. However, despite a suboptimal factor architectures, FEA is still competitive with both FEA-H and both versions of CPSO.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we proved that the full global solution $\boldsymbol{G}$ found by FEA will converge to a single point if the individual factors also converge. Even so, we also proved that FEA is still susceptible to pseudominima. Despite the fact that FEA can become stuck at pseudominima, we demonstrated that, when using certain factor architectures, the probability of factors becoming stuck at pseudominimum approaches zero. To test this, we compared hybrid versions of CPSO and FEA and demonstrated that FEA-H did not provide significant performance gains on discrete problems. Additionally, we showed that over time, FEA has a lower probability of pseudominimum than CPSO.

There are a variety of areas we wish to explore for future work. The first is further investigation into the different factor architectures for benchmark functions. While previous work by Strasser *et al.* showed the Simple Center factor architecture has consistent performance, the results comparing FEA and FEA-H suggest that it may not be the best architecture for all functions [27]. Strasser *et al.* were also able to show that the Markov architecture was the best for abductive inference in Bayesian networks. One possible way to derive better factor architectures for the benchmark functions would be to map the functions to a Bayesian network and then use the resulting Markov blankets as a way to derive factors for FEA.

Another area of research is a more in depth analysis of FEA's different parameters, such as the number of iterations during the update step. These different parameters could also affect the complexity of FEA. As demonstrated by Strasser *et al*, FEA requires more fitness evaluations than its single-population counterparts [27]. One possible explanation is that this increase in complexity is driven by the number of iterations FEA allows each subpopulation to perform during the Update step. To verify this, we plan to vary the number of iterations allowed during the update step and compare the performance in terms of fitness and number of fitness evaluations.

Finally, we plan to apply FEA to a wider range of problems; for example, additional benchmark test functions and combinatorial optimization problems such as MaxSAT. 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.

8. REFERENCES

[1] S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, 2011.

[2] M. Elad, B. Matalon, and M. Zibulevsky. Coordinate and subspace optimization methods for linear least squares with non-quadratic regularization. *Applied and Computational Harmonic Analysis*, 23(3):346–367, 2007.

[3] 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. IEEE, 2013.

[4] N. Fortier, J. Sheppard, and S. Strasser. Learning Bayesian classifiers using overlapping swarm intelligence. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS)*, pages 1–8. IEEE, 2014.

[5] N. Fortier, J. Sheppard, and S. Strasser. Abductive inference in Bayesian networks using distributed overlapping swarm intelligence. *Soft Computing*, 19(4):981–1001, 2015.

[6] 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. ACM, 2015.

[7] 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. IEEE, 2012.

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

[9] J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[10] M. Jamil and X.-S. Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194, 2013.

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

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

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

[14] X. Li and X. Yao. Cooperatively coevolving particle swarms for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 16(2):210–224, 2012.

[15] M. Mitchell, S. Forrest, and J. H. Holland. The royal road for genetic algorithms: Fitness landscapes and ga performance. In *Proceedings of the first european conference on artificial life*, pages 245–254. Cambridge: The MIT Press, 1992.

[16] L. Panait. Theoretical convergence guarantees for cooperative coevolutionary algorithms. *Evolutionary computation*, 18(4):581–615, 2010.

[17] L. Panait, R. P. Wiegand, and S. Luke. A visual demonstration of convergence properties of cooperative coevolution. In *International Conference on Parallel Problem Solving from Nature*, pages 892–901. Springer, 2004.

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

[19] 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. IEEE, 2011.

[20] M. A. Potter and K. A. De Jong. A cooperative coevolutionary approach to function optimization. In *Parallel Problem Solving from Nature?PPSN III*, pages 249–257. Springer, 1994.

[21] M. A. Potter and K. A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[22] M. Salami and T. Hendtlass. A fast evaluation strategy for evolutionary algorithms. *Applied Soft Computing*, 2(3):156–173, 2003.

[23] N. N. Schraudolph and R. K. Belew. Dynamic parameter encoding for genetic algorithms. *Machine learning*, 9(1):9–21, 1992.

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

[25] Y.-j. Shi, H.-f. Teng, and Z.-q. Li. Cooperative co-evolutionary differential evolution for function optimization. In *Advances in natural computation*, pages 1080–1088. Springer, 2005.

[26] J. C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control.* John Wiley & Sons, 2005.

[27] S. Strasser, N. Fortier, J. Sheppard, and R. Goodman. Factored evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, PP(99):1–1, 2016.

[28] S. Strasser, R. Goodman, J. Sheppard, and S. Butcher. A new discrete particle swarm optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 53–60. ACM, 2016.

[29] F. Van Den Bergh. *An analysis of particle swarm optimizers.* PhD thesis, University of Pretoria, 2006.

[30] F. Van den Bergh and A. P. Engelbrecht. Cooperative learning in neural networks using particle swarm optimizers. *South African Computer Journal*, (26):p–84, 2000.

[31] 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.

[32] R. P. Wiegand, W. C. Liles, and K. A. De Jong. Modeling variation in cooperative coevolution using evolutionary game theory. In *FOGA*, pages 203–220, 2002.

[33] R. P. Wiegand and M. A. Potter. Robustness in cooperative coevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 369–376. ACM, 2006.

[34] Z. Yang, K. Tang, and X. Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008.