A Formal Approach to Deriving Factored Evolutionary Algorithm Architectures

Shane Strasser, John Sheppard
Gianforte School of Computing
Montana State University
Bozeman, MT 59717-3880

Stephyn Butcher
Dept of Computer Science
Johns Hopkins University
Baltimore, MD 21218-2608

Abstract—**Factored Evolutionary Algorithms (FEA) are a class of evolutionary search-based optimization algorithms that have been applied successfully to various problems, such as training neural networks and performing abductive inference in graphical models. An FEA is unique in that it factors the objective function by creating overlapping subpopulations that optimize over a subset of variables of the function. One consideration in using an FEA is determining the appropriate factor architecture, which determines the set of variables each factor will optimize. In this paper, we provide a formal method for deriving factor architectures and give theoretical justification for its use. Specifically, we utilize factor graphs of variables in probabilistic graphical models as a way to define factor architectures. We also prove how a class of problems, like maximizing NK landscapes, are equivalent to abductive inference in probabilistic graphical models. This allows us to take a factor graph architecture and apply it to NK landscapes and a set of commonly used benchmark functions. Finally, we show empirically that using the factor graph representation to derive factors for FEA provides the best performance in the majority of cases studied.**

I. Introduction

Factored Evolutionary Algorithms (FEA) are a relatively new family of Evolutionary Algorithms (EA) that create overlapping subpopulations [1]. This idea is similar to how polynomials can be decomposed into a product of factors. FEA decomposes the optimization problem into a set of subpopulations, or factors that, when put together, represent full solutions to the problem. Additionally, FEA encourages the factors to overlap, which allows the factors to compete with one another to determine state values to include in the final full solution.

FEA was presented original as an extension to Particle Swarm Optimization (PSO). While PSO have been applied successfully to a wide range of problems, it is susceptible 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 an individual's position causes the individual to have low fitness [2]. FEA helps mitigate "two steps forward and one step back" in PSO by creating subpopulations that optimize over subsets of variables in the optimization problem.

One open area of research with FEA is how to determine the overlapping factors. For example, Ganesan Pillai and Sheppard created a factor for each unique path of a neural network starting and ending at an input and output node, respectively [3]. Later work by Fortier *et al.* used a different factor architecture where each factor learned the weights for each neuron in the neural network [4]. Subsequent to this work, Fortier *et al.* created subswarms for working with Bayesian networks by using a node's Markov Blanket, which consists of the node's parents, children, and children's parents [5]. In a Bayesian network, a node is conditionally independent of all other nodes in the network given its Markov Blanket. By creating subswarms consisting of a node and its Markov blanket, subswarms are able to optimize a single node in the network without having to worry about the influence from other nodes in the network.

Previous work by Strasser *et al.* demonstrated on a variety of problems that the choice of factor architecture has a strong influence on performance [1]. However, that work lacked any formal framework for deriving a factor architecture for any problem, thus requiring a user to compare different factor architectures through tuning. In this paper, we propose a general process for deriving factor architectures that can be extended to multiple types of optimization problems. In particular, our process extends an idea inspired by Fortier *et al.* [5], [6] who derived their factors using Bayesian network Markov blankets. For our approach, we derive the factor architectures by extracting the Markov blankets of each variable, but this time using a factor graph derived from characteristics of the target objective function. Factor graphs are a generalization of Bayesian networks in that they use a hypergraph representation to encode a joint probability distribution over a set of random variables. As a way to justify this approach, we begin by proving that optimizing NK-landscapes can be mapped to factor graphs, thus further strengthening the hypothesis that such mappings exist for many types of optimization problems.

Next, we investigate why certain factor architectures outperform others. Our previous work only hypothesized that identifying conditional dependence properties among the variables in a problem could be exploited to capture relevant variable interactions when defining factors since Markov blankets define global independence properties in a problem domain. In this study, we perform additional experiments to explore how factors derived in this way improve the balance of exploration and exploitation over using alternative architectures by examining convergence rate and diversity in the swarm. We show that the factor graph-based approach improves on both when compared to alternative architectures.

Finally, our previous work on PSO-based FEA used the Discrete Multi-Valued PSO (DMVPSO) algorithm as the underlying optimization algorithm [7]. However, recent work by Strasser *et al.* proposed a new discrete PSO algorithm called Integer and Categorical PSO (ICPSO) that was shown to outperform DMVPSO and other discrete PSO variants [8]. In this work, we replace DMVPSO with ICPSO and show a marked improvement over results in our prior experiments.

II. Related Work

The earliest version of FEA was called the Particle-based Routing with Overlapping Swarms for Energy Efficiency (PROSE) algorithm and was developed by Haberman and Sheppard [9]. PROSE is a version of FEA that uses PSO as the underlying optimization algorithm and was first used as a method to develop energy-aware routing protocols for sensor networks that ensure reliable path selection while minimizing energy consumption during message transmission. In their work, the authors let each subswarm represent a sensor node and all of the sensor's neighbors in the network. The authors were able to show that PROSE extends the life of the sensor networks by performing significantly better than energy-aware routing protocols that were state-of-the-art at the time.

Later, Ganesan Pillai and Sheppard developed an extension of PROSE, called Overlapping Swarm Intelligence (OSI), to learn the weights of deep artificial neural networks [3]. In that work, each swarm represents a unique path starting at an input node and ending at each output node. A common vector of weights is also maintained across all swarms to describe a global view of the network, which is created by combining the weights of the best particles in each of the swarms. The authors showed that OSI outperformed several other PSO-based algorithms as well as standard backpropagation on deep networks.

OSI has also been used for a variety of tasks with Bayesian networks, such as abductive inference, learning Bayesian classifiers, and learning latent variable parameters. Fortier *et al.* applied OSI to perform full and partial abductive inference in Bayesian networks and were able to show that OSI outperformed several other population-based and traditional inference algorithms [5], [6]. Later work by 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 [10]. The authors were able to show that, in most cases, OSI was able to outperform the competing approaches significantly. Fortier *et al.* also proposed an OSI algorithm to learn the conditional probability tables of latent variables in Bayesian networks [11]. A subswarm was created for each node with unlearned parameters and all of the variables in that node's Markov blanket. The authors demonstrated that OSI outperformed the competing approaches and that the amount of overlap between the subswarms can impact the performance of OSI.

NK fitness landscapes were first proposed by Kaufman and have been used extensively in evaluating evolutionary algorithms [12]. We too will be using these landscapes in our evaluation of the various factor architectures. Weinberger later proved that optimizing an NK landscape is NP-complete [13]. NK landscapes have been extended to allow for mixed variables, such as continuous, integer, and nominal [14]. The authors first extended the binary NK landscape to continuous by mapping it to an N-dimensional hypercube $[0, 1]^N$ and mapping values on the interior of the hypercube using a multi-linear interpolation technique. For integers, values are first mapped to $[0, 1]$ and then to a continuous NK landscape. Nominal values required the landscapes to define fitness tables that allow L values for the discrete variables.

Estimation of Distribution Algorithms (EDA) are a class of EAs that use a probability distribution to generate new individuals. The algorithm uses a set of selected individuals to generate an estimated distribution that is then used to generate new individuals [15]. EDAs are related to our work here, where we show how a factor graph can represent NK landscapes, because EDAs are able to capture the structure of variable interactions [16]. However, our work differs in that we create an exact mapping from NK landscapes to factor graphs and therefore do not have to learn a probability distribution to generate individuals in the population or swarms. This is also similar to linkage discovering and factorization algorithms presented by Wright and Sandeep; however, in our case, we already know how the different variables in the function interact with one another [17].

III. Factored Evolutionary Algorithms

Factored Evolutionary Algorithms (FEA) are a new class of optimization algorithms that work by subdividing the optimization problem. FEA is similar to the cooperative EAs like Cooperative Coevolutionary Evolutionary Algorithms (CCEA) and Cooperative Particle Swarm Optimization (CPSO); however, FEA encourages subpopulations to overlap with one another, allowing the subpopulations to compete and share information. FEA is also a generalization of OSI since it allows for any EA to be used as the underlying optimization algorithm. This allows for FEA to be a general class of algorithm that includes CCPSO, CCGA, OSI, and island EAs. Here, we present a general definition of the FEA model presented by Strasser *et al.* [1].

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

Given a function $f : \mathbb{R}^n \to \mathbb{R}$ to be optimized with parameters $\mathbf{X} = \langle X_1, X_2, \ldots, X_n \rangle$, let $\mathbf{S}_i$ be a subset of $\mathbf{X}$ of size k. We call $\mathbf{S}_i$ a *factor* but use this notation to associate each factor with a subpopulation. 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. A local subpopulation is then defined over the variables in $\mathbf{S}_i$ that are optimizing f. FEA is the case where there are factors that are proper subsets of $\mathbf{X}$ and at least one factor overlaps with another factor. Without loss of generality, assume every factor overlaps some other factor.

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

The goal of competition in FEA is to find the subpopulations with the state assignments that have the best fitness for each variable. FEA accomplishes this by constructing a global solution vector $\mathbf{G} = \langle X_1, X_2, \ldots, X_n \rangle$ that evaluates the optimized values from subpopulations. While there are many different ways to construct $\mathbf{G}$, we use the greedy approach presented by Strasser *et al.* [1].

The sharing step serves two purposes. The first is that it allows overlapping subpopulations to inject their current knowledge into one another. Previous work by Fortier *et al.* discovered that this is one of the largest contributors to the FEA's performance [5]. The second purpose of the sharing step is to set each factor's $\mathbf{R}_i$ values so that each subpopulation $\mathbf{S}_i$ can evaluate its partial solution on f. Here, we use the share algorithm presented by Strasser *et al.* where each $\mathbf{S}_i$ is seeded with a solution from $\mathbf{G}$ [1].

The entire FEA algorithm works as follows. First, all of the subpopulations $\mathbf{S}_i$ are initialized according to the optimization algorithm being used and the factor architecture. After initialization, FEA updates the states of the search by iterating over each subpopulation and having each optimize over its variables for a set number of iterations. Next, the full global solution $\mathbf{G}$ is updated by the compete algorithm. Finally, sharing is performed by seeding each subpopulation with values in $\mathbf{G}$. These three steps (update, compete, and share) are repeated until some stopping criterion is satisfied, upon which the full global solution $\mathbf{G}$ is returned as the final solution.

IV. Mapping Problems to Factor Graphs

In this work, we use FEA to maximize NK landscapes, perform abductive inference in Bayesian networks, and minimize a set of benchmark functions. To do so, we first develop a means to derive factor architectures by showing the relationship between these different problems and factor graphs. Here we prove that the problem of finding the global maximum in an NK landscape reduces to the problem of finding the most probable explanation in a factor graph. This allows us to use the resulting resulting factor graph as a means to derive factor architectures. This work is similar to that of Gao and Culberson, but differs in that we fully describe a transformation from NK landscapes to factor graphs and no learning of the Bayesian network is required [18]. It is also closely related to the work Mühlenbein and Mahnig with the key difference being that our work proves mathematically how optimizing an NK landscape reduces to abductive inference in factor graphs [19]. The algorithm by Mühlenbein and Mahnig used individuals in the EDA to estimate the parameters for the Bayesian networks whereas our proof uses the fitness functions in the NK landscape to derive the parameters for the factor graph [19]. We begin by presenting a definition of factor graphs, abductive inference for factor graphs from Abbeel *et al.*, and NK landscapes from Kauffman [12], [20].

A factor graph is representation of a joint probability distribution. Let $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ be the set of vertices and edges in the graph where $\mathbf{V} = \{\mathbf{X} \cup \boldsymbol{\phi}\}$. $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ is the set of variables and each X_i is a random variable. $\boldsymbol{\phi}$ is the set of factors that define the potentials between variables. An edge $e_{i,j} \in \mathbf{E}$ connects a factor ϕ_i to a variable X_j if X_j is an argument of factor ϕ_i. Note that edges only exist between variable nodes and factor nodes. The set of variables that are connected to factor ϕ_i is denoted as $\mathbf{D}_i$, thus we can think of each factor ϕ_i as a hyperedge in a hypergraph. A joint distribution P_Φ for a factor graph is then defined as

$$P_\Phi(X_1, \ldots, X_n) = \frac{1}{Z} \prod_{i=1}^{m} \phi_i(\mathbf{D}_i)$$

where Z is a normalization factor that is calculated as

$$Z = \sum_{X_i \in \mathbf{X}} \left(\prod_{i=1}^{m} \phi_i(\mathbf{D}_i) \right).$$

Given a variable X_i in a factor graph, we define the Markov blanket as the set of variables X_k that are connected to any of the factor nodes ϕ_j connected to the node X_i [20]. Next, we give a definition for abductive inference in factor graphs.

Definition 1. *Given a factor graph $\mathcal{G}$, a set of evidence variables $\mathbf{X}_O \in \mathcal{G}$, and their corresponding states $\mathbf{x}_O$, abductive inference is the problem of finding the maximum* a posteriori *(MAP) probability states of the remaining variables of $\mathcal{G}$, $\mathbf{X}_U = \mathbf{X} \backslash \mathbf{X}_O$. Formally, we seek to find*

$$MAP(\mathbf{X}_U, \mathbf{X}_O) = \underset{\mathbf{x} \in \mathbf{X}_U}{\arg\max} \sum_{i=1}^{m} \log \phi_i(\mathbf{d}_i). \qquad (1)$$

where $\mathbf{d}_i$ is equal to the (full or partial) state assignments for the variables connected to the factor ϕ_i defined by $\mathbf{x}_O$.

Finally, for the sake of completeness, we give a definition for NK landscapes.

Definition 2. *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 that a bit is dependent on. Given a landscape, the fitness value is calculated as*

$$f(\boldsymbol{X}) = \frac{1}{N} \sum_{i=1}^{N} f_i(X_i, nb_K(X_i)) \qquad (2)$$

where $nb_K(X_i)$ returns the K bits that are located within X_i's neighborhood. The individual functions are defined as $f_i : \mathcal{B}^K \to \mathbb{R}^+$.

Note that there are multiple ways to define the neighborhood function. In our work, we return the next K contiguous bits of the string starting at X_i. If the end of the string is reached, then the neighborhood wraps back around to the beginning of the string. Increasing K makes the variables more dependent on one another, resulting in more rugged landscapes [21].

Given all of the previous definitions, we now show how optimizing NK landscapes can be reduced to abductive inference in factor graphs.

Theorem 1. *Given an NK landscape, a corresponding factor graph can be defined such that each factor $\phi_i \in \mathcal{G}$ correspond to a function f_i from the landscape and the optimization problem in the NK landscape is equivalent to abductive inference over the factor graph $\mathcal{G}$.*

Proof. Assume we are given an NK landscape L with N bits and K interactions per bit. Remember the task for NK landscape is to find state assignments $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle$ for $\mathbf{X}$ that maximize the function given in Equation (2). This maximization problem can be formulated as

$$M(L) = \underset{x \in Val(\mathbf{X})}{\arg\max} \sum_{i=1}^{n} f_i(x_i, nb_K(x_i)).$$

We will construct a factor graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ such that the vector defining the most probable explanation for $\mathcal{G}$ is equal to the vector denoting the global maximum of L.

For each bit in L, a variable node X_i and a factor node ϕ_i are inserted into the factor graph $\mathcal{G}$. Edges are then added between ϕ_i and each node X_j in $nb_K(X_i)$. Due to our factor graph construction, $\mathbf{D}_i = \{X_i, nb_K(X_i)\}$. We then parameterize each factor ϕ_i as $\phi_i(\mathbf{D}_i) \leftarrow \exp[f_i(X_i, nb_K(X_i))]$ for each entry in the factor table. Note that if given an empty evidence set $\mathbf{X}_O$, performing abductive inference on the newly constructed factor graph (Equation 1) can be reduced to

$$MAP(\mathbf{X}_U) = \underset{\mathbf{x} \in Val(\mathbf{X}_U)}{\arg\max} \sum_{i=1}^{m} \log \phi_i(\mathbf{D}_i).$$

Since we defined $\phi_i(\mathbf{D}_i) = \exp[f_i(X_i, nb_K(X_i))]$, we have $\log \phi_i(\mathbf{D}_i) = f_i(x_i, nb_K(x_i))$. Therefore the optimization problem can be rewritten as

$$MAP(\mathbf{X}_U) = \underset{x \in Val(\boldsymbol{X})}{\arg\max} \sum_{i}^{n} f_i(x_i, nb_K(x_i)).$$

This equation is identical to the equation maximizing NK landscapes, therefore, $M(L) = MAP(\mathbf{X})$. $\square$

The result of proving Theorem 1 has several consequences. First this shows a direct relationship between NK landscapes and probabilistic graphical models, allowing us to use any results for analyzing probabilistic graphical models on NK landscapes and vice versa. For example, this result can be used as an alternative reduction to prove the NP-completeness of abductive inference in factor graphs. This includes results using FEA to perform abductive inference on Bayesian networks. Secondly, it can allow for a new way to define NK landscapes. For example, K could be viewed as an upperbound to the number of variables connected to a factor. Additionally, the discrete factors in the factor graph could be replaced with continuous factors, which would define a new class of continuous NK landscapes. For the purposes of this paper, we will restrict our use of Theorem 1 to informing us on how to apply FEA to binary NK landscapes.

We also extend this to any maximization function with a similar form. This is expressed as follows.

Theorem 2. *Given a function to be maximized with the form*

$$f(\mathbf{X}) = \sum_{i=1}^{N} f_i(X_i, X_{i+1}, \ldots, X_{i+k_i})$$

a corresponding factor graph can be defined such that optimizing the function is identical to abductive inference of the factor graph.

Proof. Assume we are given a function f to be maximized with N variables and that this maximization problem is formulated as

$$M(f) = \underset{x \in Val(\mathbf{X})}{\arg\max} \sum_{i=1}^{n} f_i(x_i, x_{i+1}, \ldots x_{i+k_i}).$$

We will construct a factor graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ such that the vector defining the most probable explanation for $\mathcal{G}$ is equal to the vector denoting the global maximum of f.

For each dimension in f, a variable node X_i and a factor node ϕ_i are inserted into the factor graph $\mathcal{G}$. Edges are then added between ϕ_i and each node X_j in $\{X_{i+1}, \ldots X_{i+k_i}\}$. Due to our factor graph construction, $\mathbf{D}_i = \{X_i, X_{i+1}, \ldots X_{i+k_i}\}$. We then parameterize each factor ϕ_i as

$$\phi_i(\mathbf{D}_i) \leftarrow \exp[] f_i(X_i, X_{i+1}, \ldots X_{i+k_i})]$$

for each entry in the factor table. Note that if given an empty evidence set $\mathbf{X}_O$, performing abductive inference on the newly constructed factor graph (Equation 1) can be reduced to

$$MAP(\mathbf{X}_U) = \underset{\mathbf{x} \in Val(\mathbf{X}_U)}{\arg\max} \sum_{i=1}^{m} \log \phi_i(\mathbf{D}_i).$$

Since we defined $\phi_i(\mathbf{D}_i) = \exp[f_i(X_i, X_{i+1}, \ldots X_{i+k_i})]$, we have $\log \phi_i(\mathbf{D}_i) = f_i(x_i, x_{i+1}, \ldots, x_{i+k_i})$. Therefore the optimization problem can be rewritten as

$$MAP(\mathbf{X}_U) = \underset{x \in Val(\mathbf{X})}{\arg\max} \sum_{i}^{n} f_i(x_i, x_{i+1}, \ldots x_{i+k_i})).$$

This equation is identical to maximizing the function f, therefore, $M(f) = MAP(\mathbf{X})$. $\square$

We note two major differences between Theorem 1 and Theorem 2. The first is that Theorem 2 does not place any restriction on the variables being continuous or discrete. Secondly, Theorem 2 does not require that each f_i use the same $k + 1$ inputs that NK landscapes require.

Note that this theorem only applies to maximization. However, the following Corollary applies the result to minimization as well.

Corollary 1. *Given a function to be minimized with the form*

$$f(\boldsymbol{X}) = \sum_{i=1}^{N} f_i(X_i, X_{i+1}, \ldots, X_{i+k_i}) \tag{3}$$

a corresponding factor graph can be defined such that optimizing the function is identical to abductive inference of the factor graph.

Proof. Given a problem to be minimized, f with N variables, we can transform the problem into a maximization problem by creating the new function $f'(\mathbf{X}) = -f(\mathbf{X})$. Theorem 2 can then be applied directly to the transformed function f'. □

V. Comparing FEA Architectures

In previous work, we provided initial experiments looking at possible factor architectures for FEA [1]. Here, we apply the formal results given in this paper so that we can evaluate the factor architectures derived from factor graphs on all of the problem types. We also extend these prior experiments by first incorporating the Integer and Categorical PSO by Strasser *et al.* for discrete problems [8]. Additionally, our prior results restricted the analysis to the average fitness of each architecture on only four networks. Here, we incorporate ten additional networks for performing abductive inference in Bayesian networks and look at other performance characteristics, such as population diversity, to gain a better understanding of where each architecture gains its advantage.

A. ICPSO

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 [8]. In ICPSO, a particle p's position is represented as $\mathbf{X}_p = [\mathcal{D}_{p,1}, \mathcal{D}_{p,2}, \ldots, \mathcal{D}_{p,n}]$ where each $\mathcal{D}_{p,i}$ 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 $\mathcal{D}_{p,i} = [d^a_{p,i}, d^b_{p,i}, \ldots, d^k_{p,i}]$, 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.

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

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.

$$\begin{aligned}\mathbf{V}_p &= \omega \mathbf{V}_p + U(0,\phi_1) \otimes (\mathbf{pBest} - \mathbf{X}_p) \\ &\quad + U(0,\phi_2) \otimes (\mathbf{gBest} - \mathbf{X}_p) \\ \mathbf{X}_p &= \mathbf{X}_p + \mathbf{V}_p\end{aligned}$$

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 $\mathbf{pBest_p}$ is replaced with $\mathbf{gBest}$ 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 $\mathbf{S}_p = [s_{p,1}, s_{p,2}, \ldots, s_{p,n}]$ 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, $\mathbf{P}_p$, and the sample itself, $\mathbf{S}_p$, 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.

B. Experimental Setup

In these experiments, we tested each of the architectures on three different problems: abductive inference in Bayesian networks, maximizing NK landscapes, and minimizing a set of commonly-used test functions. For the Bayesian networks, we used a set of networks from the Bayesian Network Repository [22]. NK landscapes were generated randomly using combinations of $N = 25, 40$ and $K = 2, 5, 10$. For each Bayesian network, 30 trials were performed for each factor architecture. Because NK landscapes are randomly generated, we generated 30 landscapes. Each version of FEA was then run 30 times on each landscape. On the benchmark test functions, we used the Brown, Dixon & Price, Powell Singular, Rana, and Rosenbrock. These functions were chosen because they match the form of the function shown in Equation 3, and each f_i is comprised of more than one variable. We also note that Brown, Powell Singular, and Rana supplement the results from [1].

On the NK landscapes and abductive inference, we used ICPSO as the underlying search algorithm. On the benchmark problems, we used a standard "gbest" 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, which was recommended by Strasser *et al.* [8]. Additionally, each factor contained ten individuals. These values were found to perform well for all architectures on all problems during tuning of the algorithms. We note that comparisons with single population PSOs and other approaches are omitted here because previous work by Strasser *et al.* and Strasser and Sheppard demonstrated that FEA versions of evolutionary algorithms outperformed a single population and cooperative coevolutionary versions [1], [23].

For each of the problem types, we used three different methods for deriving factor architectures based on factor graphs.

Parents— For each variable X_i, we construct a subpopulation of individuals consisting of elements in the neighborhood of X_i.

Markov— This architecture uses the Markov blanket of the nodes to create subpopulations, which offers arguably one of the most natural ways to subdivide a probabilistic graphical model and provide overlap. For each problem, a factor is created for each variable X_i consisting of X_i and all the nodes in X_i's Markov blanket.

Random— Random subpopulations are considered as the baseline architecture. For this approach, a random subpopulation is constructed for each of the N variables. K variables are then added to each of the N subpopulations. We used two different values for K. The first is setting it equal to the number of variables in a node's neighborhood and denote it as Random (P). Random (M) denotes a random architecture where M is set equal to the average size of a Markov blanket for each test problem.

Similar to the work by Strasser *et al.* and Fortier *et al.*, we hypothesize that the Markov architecture will outperform the other architectures [1], [5]. Additionally, we hypothesize that in the Random architectures, the subpopulations with fewer variables, Random (P), will outperform Random (M) because it will be less susceptible to "two steps forward and one step back".

C. Results

Table I shows the results for performing abductive inference on the Bayesian networks using FEA-ICPSO. Results are reported for each of the four different factor architectures in terms of average fitness values. The standard error for confidence bounds is given in parentheses. Similarly, Table II displays the results of FEA-ICPSO to maximize NK landscapes. Finally, we present the benchmark results in Table III. All results are averaged over 30 trials. In all tables, a bold value indicates that a Paired Student t-Test with $\alpha = 0.05$ determined that factor architecture significantly outperformed the others on that specific problem. All cases satisfied the requirements for the Paired Student t-Test. If multiple values in a single row are bold, that means that the bolded architectures were not statistically significantly different from each other but significantly outperformed all non-bolded architectures.

Looking at Table I, we can see that Markov always significantly outperforms all competing architectures. The next best performing architecture was Parents, as it outperformed both Random architectures on the Alarm, Andes, Child, Diabetes, Hepar2, Insurance, Link, Pathfinder, and Win95pts networks. On the Barley, Water, and Pigs networks, the Random (M) architecture performed second best, while on Mildew and Hailfinder, Random (P) was second.

For the NK landscapes, Markov again demonstrated the best performance to a statistically significant level. Out of the remaining architectures, Parents performed best and was only outperformed by Random(M) on $N = 25$ and $K = 10$.

Finally, Markov had the best overall performance on the benchmark functions, statistically outperforming the other methods in most cases. However, it failed to differ statistically from Random (M) on Dixon & Price and appeared to be outperformed by Random (P) on the Rosenbrock function. However, this difference between Random (P) and Markov on the Rosenbrock was also not significant. Parents was second only on Brown while Random (M) performed best on the Dixon & Price and second best on the Powell Singular functions. Random (P) was best on Rosenbrock and second best on the Rana.

D. Analysis

Based on our results, the Markov architecture performs best on the majority of problems. This is because this architecture groups variables together that are highly related, allowing interactions to be captured by inter-swarm optimization. Similarly the Parents architecture groups variables that are highly related; however, the resulting swarm for a variable X_i contains the variables that X_i is dependent on, but does not contain variables that depend on it. Thus, Markov superior performance can be attributed to the fact that it includes both sets of variables in each swarm, and thus does a better job of grouping highly related variables.

Even though the Markov architecture has larger factors, it does not appear to suffer from "two steps forward and one step back", which often affects populations that optimize over large sets of variables. This is illustrated by the fact Random (M), which has larger factors, outperformed Random (P) on all networks except Hailfinder, Hepar2, Mildew, and Win95pts. If "two steps forward and one step back" had been present, we would have expected the larger factors to impede the algorithm's performance. The NK landscape results further support these claims: the Random (M) architecture outperformed the smaller random architecture, Random (P), on all landscapes tested.

Additionally, Parents outperformed Random (M) on almost all NK Landscapes. This suggests that the level of interaction between factor variables, rather than the sizes of the factors, is more important when creating a factor architecture.

To investigate further where the Markov architecture's performance gains originate, we analyzed the diversity over time for each of the different architectures. Because most individuals optimize over different subsets of variables, we used the

TABLE I
RESULTS OF FEA-ICPSO PERFORMING ABDUCTIVE INFERENCE ON BAYESIAN NETWORKS.

	Markov	Parents	Random (M)	Random (P)
Alarm	**−1.01E+01 (5.74E−01)**	−1.32E+01 (6.24E−01)	−1.40E+01 (7.76E−01)	−1.55E+01 (8.49E−01)
Andes	**−5.54E+01 (5.07E−01)**	−5.98E+01 (6.72E−01)	−6.62E+01 (7.12E−01)	−6.68E+01 (1.16E+00)
Barley	**−4.08E+01 (7.76E−01)**	−4.50E+01 (1.08E+00)	−4.39E+01 (1.14E+00)	−4.94E+01 (1.42E+00)
Child	**−6.09E+00 (2.68E−01)**	−7.30E+00 (3.25E−01)	−7.71E+00 (2.93E−01)	−8.36E+00 (3.33E−01)
Diabetes	**−1.10E+04 (3.13E+02)**	−1.37E+04 (4.02E+02)	−1.43E+04 (2.13E+02)	−1.48E+04 (4.54E+02)
Hailfinder	**−2.90E+01 (3.40E−01)**	−7.03E+01 (2.76E+01)	−9.10E+01 (3.32E+01)	−5.11E+01 (1.98E+01)
Hepar2	**−1.62E+01 (4.59E−01)**	−1.71E+01 (4.82E−01)	−1.75E+01 (5.29E−01)	−1.71E+01 (5.08E−01)
Insurance	**−1.01E+01 (3.51E−01)**	−1.14E+01 (5.21E−01)	−1.16E+01 (4.03E−01)	−1.24E+01 (4.95E−01)
Link	**−3.13E+03 (2.31E+02)**	−4.76E+03 (3.96E+02)	−4.67E+03 (3.42E+02)	−6.52E+03 (3.61E+02)
Mildew	**−8.20E+02 (7.71E+01)**	−1.04E+03 (8.13E+01)	−1.06E+03 (1.03E+02)	−1.02E+03 (1.32E+02)
Pathfinder	**−4.22E+02 (5.94E+01)**	−7.03E+02 (7.91E+01)	−1.14E+03 (1.02E+02)	−1.34E+03 (1.47E+02)
Pigs	**−2.15E+02 (9.34E−01)**	−2.22E+02 (1.19E+00)	−2.21E+02 (1.11E+00)	−2.43E+02 (1.95E+01)
Water	**−3.27E+02 (5.52E+01)**	−4.67E+02 (6.80E+01)	−3.87E+02 (8.32E+01)	−5.45E+02 (8.25E+01)
Win95pts	**−1.76E+01 (6.68E−01)**	−2.32E+01 (8.08E−01)	−5.04E+01 (1.96E+01)	−3.16E+01 (1.17E+00)

TABLE II
RESULTS OF FEA-ICPSO MAXIMIZING NK LANDSCAPES.

		Markov	Parents	Random (M)	Random (P)
N = 25	K = 2	**1.78E+01 (2.21E-02)**	1.76E+01 (2.42E-02)	1.72E+01 (2.68E-02)	1.71E+01 (2.84E-02)
	K = 5	**1.81E+01 (1.64E-02)**	1.79E+01 (1.92E-02)	1.77E+01 (1.92E-02)	1.72E+01 (2.20E-02)
	K = 10	**1.78E+01 (1.62E-02)**	1.75E+01 (1.64E-02)	1.77E+01 (1.47E-02)	1.72E+01 (1.87E-02)
N = 40	K = 2	**2.81E+01 (3.07E-02)**	2.78E+01 (3.13E-02)	2.72E+01 (3.53E-02)	2.71E+01 (3.55E-02)
	K = 5	**2.89E+01 (2.22E-02)**	2.84E+01 (2.43E-02)	2.79E+01 (2.77E-02)	2.73E+01 (2.97E-02)
	K = 10	**2.84E+01 (2.13E-02)**	2.81E+01 (2.15E-02)	2.78E+01 (2.33E-02)	2.72E+01 (2.47E-02)

TABLE III
RESULTS OF FEA-PSO MINIMIZING BENCHMARK FUNCTIONS.

	Markov	Parents	Random (M)	Random (P)
Brown	**1.66E−02 (1.51E−03)**	3.39E−02 (3.20E−03)	4.12E−02 (6.99E−03)	6.09E−02 (6.23E−03)
Dixon & Price	**3.60E+00 (4.22E−01)**	4.10E+00 (3.55E−01)	**3.57E+00 (5.64E−01)**	5.64E+00 (5.75E−01)
Powell Singular	**4.76E−01 (2.74E−02)**	6.68E−01 (5.05E−02)	6.44E−01 (5.04E−02)	4.35E+00 (3.26E+00)
Rana	**−1.39E+04 (9.46E+01)**	−1.32E+04 (1.29E+02)	−1.30E+04 (1.28E+02)	−1.24E+04 (1.30E+02)
Rosenbrock	6.92E+01 (9.01E+00)	6.94E+01 (1.28E+01)	5.66E+01 (1.19E+01)	7.15E+01 (6.27E+00)

genotype variance, a measure of the distance between each individual and the "average" individual, to measure diversity [24]. This variance is calculated as $\frac{1}{N\times P}\sum_{i=1}^{N}\sum_{j=0}^{P}(\bar{x}_i - x_{i,j})^2$ where $\bar{x}_i$ is X_i average value across the population for variable, $x_{i,j}$ is individual i's value for variable X_j, P is the total number of individuals, and N is the number of variables in the problem being optimized. The graphs for both fitness and diversity over time for FEA-PSO are shown in Figure 1.

Results are presented for 100 trials of FEA-ICPSO optimizing an NK landscape with parameters $N = 25$ and $K = 2$. The y-axis on the left denotes the average diversity, while the y-axis on the right is the best fitness. The x-axis shows the number of FEA iterations, where each iteration consists of the update, compete, and share steps, and each factor performs 5 updates during a single iteration.

When considering fitness, the Markov architecture converges the fastest, and it maintains the best population diversity over time. We believe this is because the Markov architecture provides the best balance between the number of variables in the factors and the level of interaction being handled. Larger factors provide more variation between individuals' state assignments, thus leading to higher diversity in the

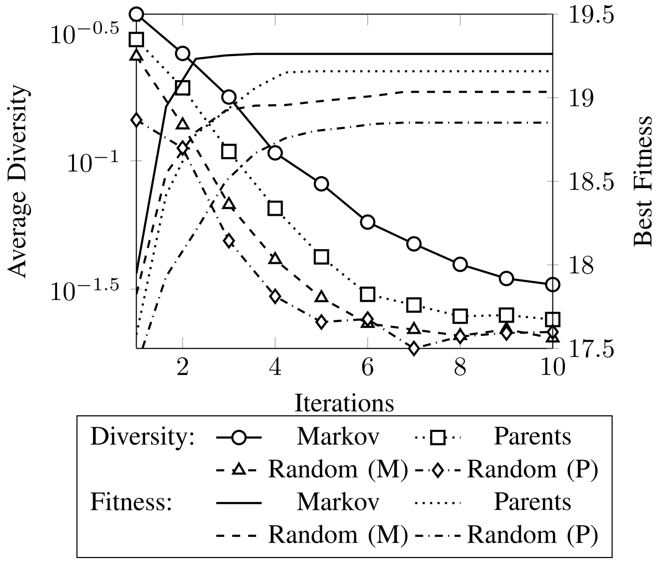

Fig. 1. Average diversity and best fitness of all individuals for NK landscape N = 25, K = 2.

population. Given this, we might expect the Markov and Random (M) architectures to have identical diversity curves, as they yield the same factor sizes for NK Landscapes; however, we observed that both Random architectures had relatively low diversity. Also, while Parents has smaller factors than Random (M), its diversity was higher on average. This implies that factor size is not the only influence on average diversity, and that grouping together highly interactive variables also increases the average diversity.

VI. Conclusions and Future Work

We present a formal method for deriving factor architectures for FEA. First, we provide a mapping of the optimization problems to a factor graph. Based on the resulting factor graph, a factor architecture derived. Finally, we demonstrate that for the majority of problems studied, the optimal architecture is based on Markov blankets in the factor graph.

Several different areas for future work exist. First we plan to investigate why the Markov architecture performed worse than the Random architectures on Rosenbrock. One hypothesis is that the relationships between variables in the Markov blanket are actually deceptive, causing FEA to become trapped in a local minimum.

Second, we plan to explore the convergence properties of FEA. To accomplish this, we will examine convergence properties for various EA methods, such as PSO, GA, and CCGA, and examine whether they can be adapted to FEA. This will extend our recent work that provided a more generalized view of FEA, independent of the underlying algorithms [23]. We will also investigate how FEA's parameters, such as the number of iterations during inter-factor and intra-factor optimization affect FEA's performance.

In this paper, we assumed that a factor graph could be derived by analysis of the actual fitness function. However, there may be cases where the fitness function will not be directly observable. For those situations, we will examine first sampling the fitness function and then using the generated samples to generate a factor graph. The resulting factor graph can then be used to generate the factors for FEA.

A more dynamic approach would be to adapt FEA to use algorithms like EDA and linkage analysis, where during the update of the factors, FEA would use one of these methods to determine if variables needed to be added or removed from a factor. This would result in an FEA that automatically detects variable interactions and adjusts accordingly.

Finally, we plan to investigate the scalability of FEA. Previous work by Strasser *et al.* demonstrated that FEA can require a large number of fitness evaluations due to the competition step [1]. We will explore how to reduce complexity by using different competition schemes.

References

[1] S. Strasser, J. Sheppard, N. Fortier, and R. Goodman, "Factored evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 2, pp. 281–293, 2017.

[2] F. Van den Bergh and A. P. Engelbrecht, "Cooperative learning in neural networks using particle swarm optimizers," *South African Computer Journal*, no. 26, pp. p–84, 2000.

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

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

[5] N. Fortier, J. Sheppard, and S. Strasser, "Abductive inference in bayesian networks using distributed overlapping swarm intelligence," *Soft Computing*, vol. 19, no. 4, pp. 981–1001, 2015.

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

[7] 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)*, 2007, pp. 141–149.

[8] 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)*. ACM, 2016, pp. 53–60.

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

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

[11] ——, "Parameter estimation in Bayesian networks using overlapping swarm intelligence," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 2015, pp. 9–16.

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

[13] E. D. Weinberger, "NP completeness of Kauffman's NK model, a tuneable rugged fitness landscape," Tech. Rep., 1996.

[14] R. Li, M. T. Emmerich, J. Eggermont, E. G. Bovenkamp, T. Bäck, J. Dijkstra, and J. H. Reiber, "Mixed-integer NK landscapes," in *Parallel Problem Solving from Nature-PPSN IX*. Springer, 2006, pp. 42–51.

[15] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz, "Linkage problem, distribution estimation, and Bayesian networks," *Evolutionary Computation*, vol. 8, no. 3, pp. 311–340, 2000.

[16] Q. Zhang and H. Mühlenbein, "On the convergence of a class of estimation of distribution algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, pp. 127–136, 2004.

[17] A. H. Wright and S. Pulavarty, "On the convergence of an estimation of distribution algorithm based on linkage discovery and factorization," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 2005, pp. 695–702.

[18] Y. Gao and J. Culberson, "On the treewidth of NK landscapes," in *Proceedings of the Genetic and Evolutionary Computation (GECCO)*. Springer, 2003, pp. 948–954.

[19] H. Mühlenbein, T. Mahnig, and G.-F. Informationstechnik, "Convergence theory and applications of the factorized distribution algorithm," *Journal of Computing and Information Theory*, vol. 7, no. 1, pp. 19–32, 1999.

[20] P. Abbeel, D. Koller, and A. Y. Ng, "Learning factor graphs in polynomial time and sample complexity," *The Journal of Machine Learning Research*, vol. 7, pp. 1743–1788, 2006.

[21] H. Aguirre and K. Tanaka, "A study on the behavior of genetic algorithms on NK-landscapes: Effects of selection, drift, mutation, and recombination," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 86, no. 9, pp. 2270–2279, 2003.

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

[23] S. Strasser and J. W. Sheppard, "Convergence of factored evolutionary algorithms," in *Proceedings of the Conference on Foundations of Genetic Algorithms (FOGA) XIV*. ACM, 2017, pp. 81–94.

[24] K. Matsui, "New selection method to improve the population diversity in genetic algorithms," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, vol. 1, 1999, pp. 625–630.