ABSTRACT
We propose a new estimation of distribution algorithm called Factor Graph based Factorization Distribution Algorithm (FG-FDA). In FG-FDA, a factor graph is used to encode the underlying distribution of the problem. In order to learn the factor graph from a population of potential solutions, a symmetric non-negative matrix factorization is employed to factorize the matrix of pair-wise dependencies between variables of the problem. To show the performance of the FG-FDA, encouraging experimental results on different problems are provided as the support to the mathematical analysis of the approach. The experiments show that FG-FDA is capable of solving the optimization problems in polynomial time with polynomial number of evaluations.

GECCO track: Estimation of Distribution Algorithm.

Categories and Subject Descriptors
G.1.6 [Numerical Analysis]: Optimization

General Terms
Algorithm, Design, Experimentation, Performance

Keywords
Optimization problems, Estimation of distribution algorithms, Factor graph, Linkage learning, Mutual information, Matrix factorization

1. INTRODUCTION
In the last decade, lots of works are done on optimization algorithms capable of learning linkage along with optimization. Attention to this class of optimization solutions arise when deficiencies of simple genetic algorithm was revealed with a class of fitness functions, called deceptive functions [1]. It is showed that to be able to solve such problems, the optimization algorithms need to learn the underlying structure of the variable’s dependencies. To learn this structure, the only information available to the optimization algorithms is a population of the fittest individuals with their fitness values. Estimation of Distribution Algorithms (EDA) are one class of the solutions that learn the variable’s dependencies’ structure and overcome this deficiency of genetic algorithm. EDAs estimate the joint distribution of the multinomial data in order to generate new solutions. Encoding the joint distribution of the data is usually done using a graphical model. Bayesian networks [2] and markov models [3] are widely used as the model in the model building process of EDAs, but factor graphs which are the ideal graphical model to encode the distribution of the data are rarely used [4] possibly because there has been no efficient learning algorithm to learn this kind of graphical models.

In this paper, we introduce an efficient factor graph learning algorithm. To learn the factor graph we first change the space of the problem from learning the graphical model to the space of graph theory and matrix factorization. We first introduce a mapping between these two spaces, then explain a matrix factorization approach to learn a factor graph. After learning the factor graph, new potential solutions are produced. The process of model building and creating new samples are repeated until the optimum solution is found. Utilizing the factor graph has some advantages over other models. First and most important of all, this graphical model vividly and explicitly shows the structure of the dependencies between variables of the problem. This human-readable information can easily be used by experts to analyze the problem. The results and the revealed structure of the problem can be used for analyzing similar problems. Second, having the the factor graph and the factorization we can use this information not only for re-sampling the new potential solutions which is not an easy task in undirected graphical models, but also employing other approaches like BB-wise search and crossover to find the optimum values of each building block.

The outline of this paper is as follows: In the next section, some related works are reviewed. In section 3, factor graphs are discussed. Section 4 is all about the algorithm for learning the factor graph and its background information. Section 5 presents the complexity analysis of the algorithm. In section 6, reference algorithms are introduced. Section 7 presents the experimental results of the algorithm. To show the performance and efficiency of the proposed approach, it is tested on some benchmark problems and its effectiveness is discussed. Finally in section 8, some conclusion remarks,
that a factor node (a hyper-edge) can connect more than one variable node. As it is obvious, factor graph (or hypergraph) is the ideal graphical model to express the dependencies of the variables of a function. But learning the factor graph is not an easy task. In this paper, we have used a matrix factorization technique to learn the factor graph as the model. Then, the factor graph is used to generate new potential solutions.

In order to find the factor graph, we use a matrix factorization technique which has been mathematically proven to converge in polynomial time. To do so, we employ variable dependencies to partition the variables probabilistically. Based on the concept of random walks on graphs, in our approach, the variable dependencies are modelled as empirical transitions generated by a mixture of latent factors.

4. PROPOSED APPROACH

Variable dependencies can be encoded by an undirected graph, where vertices denote variables and edges’ weights represent variable dependencies. In the proposed approach, this graph called the Underlying Dependency Graph of the Problem (UDGP) is first constructed from a selected population of potential solutions using pair-wise dependencies of the variables. UDGP is then factorized into a bipartite graph which represents the factor graph of the problem. So in the proposed approach, we are converting pair-wise dependencies to higher-order dependencies.

The algorithm has three main steps: 1) Constructing the UDGP and making it sparse 2) Factorizing the UDGP into a factor graph 3) Generating new potential solutions based on the factors of the learned factor graph.

The approach, as will be explained here, is capable of learning disjoint and overlapped linkage groups even for problems with linkage groups of varying size. The remainder of this section describes details of the proposed approach.

Step 1: Constructing The Underlying Dependency Graph

The underlying dependency graph $UDGP(V, E)$ is a weighted, undirected graph with vertices $V = \{v_i : i \in S\}_{i=1}^n$ ($S$ is a finite set used for indexing the variables) and edges $E \subseteq \{(v_i, v_j)\}$, where each vertex corresponds to one decision variable and each edge of the graph has a weight $w_{ij} = \epsilon_{ij}$ representing the strength of dependency between vertex $i$ and vertex $j$. Edge weights $\{\epsilon_{ij}\}$ are real numbers. The larger the $\epsilon_{ij}$ is, the stronger the dependency is between vertex $i$ and vertex $j$. To construct the dependency graph, a pairwise dependency metric is calculated for each pair of variables. Mutual information, simultaneity metric [10], nonlinearity [11] and other pair-wise metrics can be used. The pair-wise metric we used is explained in the next subsection.

After constructing the UDGP, some of the too weak edges are pruned using a threshold to accelerate the next step of the algorithm. The threshold used is set to $\mu$, where $\mu$ is the mean of the UDGP.

Figure 2 shows an example of an underlying dependency graph of a problem with 12 variables with three building-blocks. The thickness of the edges shows their weight magnitude. Lines are the correct linkages and dashed lines are false linkages. Some of the edges are not shown because their weights are too weak.
If the dependency graph is correctly constructed, the edges between dependent variables would be stronger than the edges between non-dependant variables.

**Pairwise dependencies:**

There are number of pairwise metrics which have been used as the interaction metric in GAs. Nonlinearity and entropy are the two commonly used metrics. The linearity-nonlinearity check (LINC) was first introduced by Muntean and Goldberg [11]. To find the pairwise dependencies, LINC uses an evaluated population of potential solutions. It is based on investigating fitness change by perturbations in a pair of variables. In [12], it is showed that the non-linearity check (LINC) was first introduced by Muntean and Goldberg [11]. To find the pairwise dependencies, LINC uses an evaluated population of potential solutions. It is based on investigating fitness change by perturbations in a pair of variables. In [12], it is showed that the non-linearity check (LINC) was first introduced by Muntean and Goldberg [11].

Assume two random variable $X$ and $Y$ with probability mass function $P_{XY}(x,y)$, mutual information is defined as:

$$I(X;Y) = \sum_{x,y} P_{XY}(x,y) \log \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)} = E_{P_{XY}} \log \frac{P_{XY}}{P_XP_Y}. \tag{2}$$

In [12], it is showed that mutual information is a better metric in comparison to nonlinearity to be used as the interaction metric to avoid building blocks disruptions in GAs. So we use mutual information in our proposed algorithm to construct the underlying dependency graph of the problem UDGP:

$$e_{ij} = \sum_{i,j \in V} P(i,j) \log \frac{P(i,j)}{P(i)P(j)}. \tag{3}$$

**Step 2: Factorizing the UDGP into a factor graph**

Let $K(V,U,F)$ be the bipartite graph, where $V = \{v_i : i \in S\}_{i=1}^{n}$ is a collection of variables indexed by a finite set $S$ and $U = \{u_p : p\}_{p=1}^{m}$ is a set of factor nodes ($V$ and $U$ are disjoint sets) and $F$ contains all the edges connecting $V$ and $U$. Let $B = \{b_{ip}\}$ denote the $n \times m$ adjacency matrix with $b_{ip} \geq 0$ being the weight for edge $[v_i, u_p]$. To factorize the UDGP and finding the corresponding bipartite graph, we use the similarity relation between $v_i$ and $v_j$ in the bipartite graph $K$ presented in [15].

$$e_{ij} = \sum_{p=1}^{m} b_{ip}b_{jp} \lambda_p = (B\Lambda^{-1}B^T)_{ij}, \quad \Lambda = \text{diag}(\lambda_1, ..., \lambda_m) \tag{4}$$

Where $\lambda_p = \sum_{i=1}^{n} b_{ip}$ is the degree of vertex $u_p \in U$.

The above equation can be analyzed based on the random walks on graphs. $e_{ij}$ is proportional to the stationary probability of transition between $v_i$ and $v_j$, $p(v_i,v_j)$. All the paths between vertices in $V$, must go through vertices in $U$ in the factor graph, therefore:

$$p(v_i,v_j) = p(v_i)p(v_j|v_i) = d_i \sum_{p} p(u_p|v_i)p(v_j|u_p)$$

$$= \sum_{p} p(v_i,u_p)p(u_p,v_j) \lambda_p, \tag{5}$$

where $d_i = p(v_i)$ is the degree of $v_i$. When $b_{ip} = p(v_i,u_p)$, equation (4) and equation (5) are the same.

$p(u_p|v_i) = \frac{b_{ip}}{b_{ip}}$ is the conditional probability of transitions from $v_i$ to $u_p$ and indicates how likely variable $i$ belongs to factor nodes $p$. The corresponding bipartite graph of the UDGP in figure 2 is illustrated in figure 3.

Based on equation (4), the bipartite graph can be approximated by minimizing the distance $E, B\Lambda^{-1}B^T$. To make the problem easier we use $H = B\Lambda^{-1/2}$; then we have

$$\min_{H \in \mathbb{R}^{n \times m}} \text{distance}(E, HH^T), \quad \text{s.t.} \ h_{ip} \geq 1, \tag{6}$$

This problem is a symmetric non-negative matrix factorization (SNMF) [16]. There are different numerical methods to find the local minima of this problem. Here we use the gradient descent method to minimize the divergence distance between the two adjacency matrices (equation (7)).

**Figure 2:** The underlying graph of a problem with 12 variables and 3 disjoint building-blocks.

**Figure 3:** The bipartite graph of the UDGP in figure 2.
\[ DD(X, Y) = \sum_{i,j} (x_{ij} log \frac{x_{ij}}{y_{ij}} - x_{ij} + y_{ij}) \] (7)

**Theorem 4.1** The distance is non-increasing under the following update rule
\[ h_{ip} = \frac{h_{ip}}{\sum_j h_{jp}} \sum_j (H H^T)_{ij} h_{jp}. \] (8)

The distance is invariant under the update rule if and only if \( H \) is at stationary point of the distance [17].

After \( H \) is calculated, \( B \) can be calculated using \( B = HA^{1/2} \) and \( \Lambda = \text{diag}(\lambda_1, ..., \lambda_m) \) where \( \lambda_p = \sum_{i=1}^{n} h_{ip} \).

Proof of the convergence of the algorithm is available in [17]. In order to provide an algorithmic scheme look at the following pseudo-code:

**Algorithm 1** Learning the Factor Graph via SNMF

**Input:** A non-empty \([V \times |V|\) matrix \( E, maxFN \in \{1, ..., N\}\)
**Output:** factor graph
Initialise: Start with a \( n \times m \) matrix \( H \), where \( U = maxFN \) and \( H_{ij} \in \{0, 1\} \) is randomly set.
repeat
Update \( H \) based on equation (8).
until \( H \) is invariant.
Construct \( \Lambda = \text{diag}(\lambda_1, ..., \lambda_m) \) using \( \lambda_p = \sum_{i=1}^{n} h_{ip} \).
Calculate \( B \) using \( B = HA^{1/2} \).

**Step 3: BB-wise crossover and intra-BB search**

In this step, after learning the factor graphs and having the factors, the optimum value of the variables should be found. To do so, we can perform BB-wise crossover and/or we can perform a search on each building block to find the optimum.

In this paper, we perform a search on the best individual of the population. For every one of the identified BBs, the fitness of the individual with all the permutations of that BB is calculated and the value of the variables in the BB, are set to the most fitted permutation. Then BB-wise crossover is done on the population. The best individual is preserved in the population of the next iteration.

5. **COMPLEXITY ANALYSIS**

In this section, the number of function evaluations and the time complexity of the proposed optimization algorithm are discussed. As explained above, the algorithm is comprised of three main steps of (1) constructing the underlying graph, (2) learning the factor graph (3) finding the optimum using intra-BB search and BB-wise crossover.

**Number of evaluations:** In the first step of the algorithm, for constructing the UDGP, and in the third step for intra-BB search, evaluation is performed. Each individual in the population is evaluated exactly once in the first step. In the third step, for a problem with \( m \) BBs and maximum length of \( k \) for each BB, at most \( m \times 2^k \) evaluations are done. Therefore, if the population size is \( N \), number of fitness evaluations in each iteration would be \( m \times 2^k + N \) and if number of generations is \( g \), the overall number of evaluations of the algorithm would be \((m \times 2^k + N) \times g\).

**Time:** Pair-wise dependencies (mutual information) are computed by a pass through all the strings in the population. As \( MI \) values are calculated for each pair of variables therefore this computation is done in \( O(n^2 \times N) \), where \( n \) is the problem size and \( N \) is the population size.

In the process of learning the factor graph, using the divergence distance for minimizing the distance between the two adjacency matrices (equation (7)), it is only needed to sum over all non-zero terms of matrix \( E \) for each update. So, if \( E \) is sparse, the time complexity of the equation (8) is \( O(m^2 \times n \times L) \), where \( L \) is the number of nonzero entries in \( E \), \( m \) is the number of factor nodes and \( n \) is the number of variables. Therefore the overall time complexity of the algorithm is \( O(n^2 \times N + m^4 \times n \times L) \). Considering the worst case where \( L = n^2 \), time complexity of the Fg-FDA is \( O(n^2 N + m^4 n^3) \).

**Algorithm 2** Algorithm of the proposed approach

**Output:** optimum solution
1: Create random population.
2: repeat
3: Evaluate the population.
4: Select population by tournament selection.
5: for \( N \) do
6: for each variable \( i \) and variable \( j \), do
7: Calculate \( e_{ij} = MI(i, j) \)
8: end for
9: end for
10: Prune the UDGP by threshold = \( \mu_{UDGP} \).
11: Do the SNMF and finding the factor nodes using the algorithm 1.
12: Perform intra-BB search.
13: Perform BB-wise crossover.
14: Replace population.
15: until optimum found || maximum number of generations reached || population converged

6. **REFERENCE ALGORITHMS**

Two approaches are used as reference algorithms. Both are well studied optimization approaches in the category of estimation of distribution algorithms. Bayesian optimization algorithm (BOA) [2] and dependency structure matrix genetic algorithm (DSMGA) [13] are used to compare the results.

DSMGA uses mutual information to construct the dependency structure matrix and then cluster the DSM by an evolutionary strategy with a fit-to-data objective function. DSMGA then performs a BB-wise crossover to find the optimum. To be fair in the comparisons of results, the same intra-BB search is added to DSMGA as well.

BOA uses conditional probabilities of variables, a greedy algorithm and a scoring metric to learn a Bayesian network as the underlying probabilistic model of the problem. BOA then use the learned model and conditional probabilities to sample new potential solutions. For DSMGA, RTR is used to replace the population. For
BOA replacement with elitism is done. Worst half of the population is replaced with new offsprings. For the FG-FDA, replacement with elitism is used. In every generation only the best individual is preserved in the population of the next generation.

7. EXPERIMENTAL RESULTS

In this section, the experimental results are presented. First the experiments and test functions are explained and then the results are shown and analyzed.

Test Functions

An additively separable deceptive function of order $k$ called $k$-deceptive is defined as the sum of single deceptive functions of order $k$ applied to non-overlapping $k$-bit partitions of solution strings.

$$f(x) = \begin{cases} f_{High} & \text{if } u(x) = k \\ f_{Low} - u(x) \cdot f_{Low}/k & \text{else} \end{cases}$$

where $u(x)$ returns the number of 1s in string $x$. Usually $f_{High} = 1$ and $f_{Low} = 0.9$. It has one global optimum in individual of all 1s and one local optimum in individuals of all 0s. The function is difficult because the local optimum has larger basin of attraction.

A Concatenated deceptive function is a sum of deceptive subfunctions. In this paper, concatenated deceptive functions are used as the test function. A concatenated deceptive function with $m$ subfunctions, has one global optimum and $2^m - 1$ local optima.

Nearest-neighbour NK landscape

An NK fitness landscape is defined by: (1) The number of bits, $n$. (2) The number of neighbours per bit, $k$. (3) A set of $k$ neighbours $\Pi(X_i)$ for the $i$-th bit, $X_i$, for every $i \in \{0, ..., n-1\}$. (4) A sub-function $g_i$ defining a real value for each combination of values of $X_i$ and $\Pi(X_i)$ for every $i \in \{0, ..., n\}$. A look-up table with $2^{k+1}$ values define each sub-function. Maximizing the objective function (10) is the goal.

$$f_{nk}(X_0, ..., X_{n-1}) = \sum_{i=0}^{n-1} g_i(X_i, \Pi(X_i)).$$

The four components of an NK problem instance affect the difficulty of optimizing NK landscapes. Analyzing the complexity of NK landscapes can be done by focusing on the influence of $k$. For $k = 0$, NK landscapes are simple functions similar to one-max, which can be solved in linear time and therefore is an easy-GA problem. For $k = 1$, the global optimum of NK landscapes can be obtained in polynomial time, but for $k > 1$, finding the global optimum of unrestricted NK landscapes is NP-complete [18]. If the NK problem is restricted so that neighbours can only be adjacent string positions, the problem becomes polynomially solvable with dynamic programming for $k > 1$ [19]. NK landscapes have attracted researchers in stochastic optimization because of their difficulties and properties.

Setup of experiments

The maximum number of iterations for each problem instance is set to the overall number of bits in the problem. Each run is terminated either when the global optimum has been found or when the population has converged to a single solution or when the maximum number of iterations has been reached.

The results are shown for the concatenated deceptive functions with linkage groups of sizes 3 (for problem sizes 60 to 600) and 5 (for problem sizes 60 to 600). Population size is determined by bisection with 30 successful runs. The reported number of fitness evaluations are averaged over 30 independent runs.

For the NK landscapes problem, we use $k = 5$ and we consider $n$ from 20 to 100. For each $n$, we use 1000 unique, independently generated instances.

To evaluate performance of the algorithm, the overall number of fitness evaluations is reported. We have fitted our results to $N \approx am^b \log m$ reported for EDAs. Based on the coefficient of determination ($R^2$) both models are acceptable. This deduction is based only on the available data.

In figures 4 and 5, the population sizes for concatenated 3-deceptive and concatenated 5-deceptive functions are plotted with the fit model $N \approx a \times m^b \log m$. As it can be seen $b$ is in the range of $(1.2 - 1.5)$ for both problems.

Figure 4: Population size needed for the FG-FDA to successfully solve the problem for concatenated 3-deceptive function with the fit model.

Figure 5: Population size needed for the FG-FDA to successfully solve the problem for concatenated 5-deceptive function with the fit model.

In figures 6 and 7, the number of fitness evaluations for concatenated 3-deceptive and concatenated 5-deceptive are...
plotted together with the fit model $am^b \log m$. $b$ is observed to be 1.4 for the FG-FDA. As explained in section 4 an intra-BB search is done to find the optimum value of the variables of a building-block for the best individual of the population. This adds a $m \times 2^k$ evaluations in every generation of the algorithm which are counted in the overall number of evaluations.

In figure 8, number of evaluations for FG-FDA to solve nearest neighbour NK landscape problems of different sizes are plotted. As it is illustrated FG-FDA is capable of effectively solving the highly overlapped nearest neighbour NK landscape problems as well. More results on such problems and analyzing the performance of the algorithm based on different parameters are left for the future works.

**Comparison with the reference algorithms**

The number of fitness evaluations for BOA, DSMGA and FG-FDA to solve concatenated 3-deceptive and 5-deceptive are depicted in figures 9 and 10. Comparing the results visually, the FG-FDA needs smaller number of fitness evaluations and grows slower with regard to the problem size.

**8. SUMMARY AND CONCLUSION**
This paper introduces a factorization distribution algorithm. The factorization is done by learning a factor graph using a symmetric non-negative matrix factorization approach. The matrix of pair-wise dependencies between variables of the problem is used to be factorized into a factor graph. FG-FDA consists of three main steps. First, a dependency graph is created using a pairwise metric. Second, a matrix factorization approach is employed to learn a factor graph. Third, new potential solutions are created based on the factor nodes using intra-BB search and BB-wise crossover. The proposed approach uses the well studied mutual information as the pairwise dependencies between variables. To demonstrate the performance of the FG-FDA, the results on deceptive functions and NK landscape problems are reported. It is shown that FG-FDA can solve all the tested problems with polynomial number of fitness evaluations in polynomial time with respect to length of the problem. The results are compared to the results of two well-known reference approaches and it is shown that performance of the three methods is comparable in terms of the number of function evaluations. One of the advantages of the FG-FDA is that it learns the factor graph which is naturally the ideal model to encode the variable dependencies in polynomial time. The algorithm description is easy and there is no black-box part in the algorithm. The learned model is simple to understand for humans. This can be a valuable property for the experts. The learning process has strong mathematical background and it is mathematically proved to converge. It is expected that FG-FDA has the potential of solving the hierarchical problems by hierarchically factorizing the factor nodes. More analytical discussions on different problems and different parameters of the algorithm as well as experimental results on hierarchical problems are left for the future works.

9. REFERENCES


