Int. J. Appl. Math. Comput. Sci., 2014, Vol. 24, No. 3, 621–633
DOI: 10.2478/amcs-2014-0045

A FACTOR GRAPH BASED GENETIC ALGORITHM

B. HODA HELMI *, ADEL T. RAHMANI *, MARTIN PELIKAN **

* Department of Computer Engineering
Iran University of Science and Technology, Narmak, Tehran, Iran
e-mail: helmi@iust.ac.ir

**Google Inc., Mountain View, CA, USA

We propose a new linkage learning genetic algorithm called the Factor Graph based Genetic Algorithm (FGGA). In the FGGA, a factor graph is used to encode the underlying dependencies between variables 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. To show the performance of the FGGA, encouraging experimental results on different separable problems are provided as support for the mathematical analysis of the approach. The experiments show that FGGA is capable of learning linkages and solving the optimization problems in polynomial time with a polynomial number of evaluations.

Keywords: optimization problems, genetic algorithms, estimation of distribution algorithms, factor graph, matrix factorization.

1. Introduction

The importance of linkage learning in optimizing hard problems in the context of genetic algorithms was revealed a long times ago. Probabilistic Model Building GAs (PMBGAs) and/or Estimation of Distribution Algorithms (EDAs) emerged as the extension of genetic algorithms (Mühlenbein and Paaß, 1996; Pelikan *et al.*, 2002; Larrañaga and Lozano, 2001) to optimize functions that the GA (with no linkage learning capability) has difficulties in optimizing. EDAs are population based search algorithms that incorporates probability distributions to exploit the correlation of variables and generate new potential solutions. The key idea in EDAs is identification of variable dependencies and conducting the search toward the optimum, based on the information of variable dependencies.

There are many EDAs, each of which uses different approaches to learn and encode the distribution and sample new potential solutions (Mahnig and Mühlenbein, 1999; Pelikan, 2005; Sastry and Goldberg, 2000; Yu and Goldberg, 2006; Miquelez *et al.*, 2004). Most of these approaches are population based and generation search processes. Among all the methods used to learn and encode the distribution, graphical models are employed and investigated in many studies. Bayesian networks, which are acyclic directed graphical models, and Markov random fields, which are undirected and may be cyclic, are used in several EDAs. There are pros and cons for using each of these graphical models in EDAs. On the one hand, learning a Bayesian network from data is a computationally costly task (Mendiburu *et al.*, 2007); on the other, sampling from undirected graphical models is difficult and requires Gibbs sampling, which is computationally expensive (Mühlenbein, 2008).

Factor graphs are another type of undirected graphical models and subsume both Bayesian networks and Markov networks in that every Bayesian network or Markov network can be converted to its corresponding factor graph of the same size (Kschischang *et al.*, 2001). These graphical models are the most natural graphical representations for the structure of Additively Decomposable Functions (ADFs) (Kschischang *et al.*, 2001). Learning factor graphs for an ADF is more representative than learning Bayesian networks because they match the ADF structure better (Mühlenbein, 2012). A factor graph clearly shows the underlying structure of the problem and is readable for experts. This is an important property for the expert, when optimizing problems with an unknown dependency structure. But learning the structure of these kinds of

graphical models and sampling using these distributions is difficult. Abbeel *et al*. (2006) proved the first polynomial time and polynomial sample complexity structure learning algorithm and parameter learning algorithm for factor graphs, although the proposed structure learning algorithm is still exponential in the maximum factor scope size and the maximum Markov blanket size, but polynomial in the number of variables, n. Here, we introduce a factor graph structure learning approach in the context of an evolutionary algorithm, which is polynomial in the number of variables and the number of Building Blocks (BBs).

There are EDAs that utilize factor graphs to encode the distribution of the problem. Mühlenbein (2008) proposed using factor graphs as the model to represent the distribution of the problem in EDAs. He proposed using the algorithm introduced by Abbeel *et al*. (2006) to learn factor graphs and factor graph distributions based on the product of local probabilities. Santana *et al*. (2008) introduced an adaptive EDA approach. They learn the structure of the factor graph using the chi-square independence test. Then Kikuchi approximation of the distribution is used and finally Gibbs sampling is applied to sample new potential solutions. In the work of Mendiburu *et al*. (2007), to search for the optimum, Loopy Belief Propagation (LBF) is used to find the most probable configuration of the distribution. They first learn a Bayesian network from a population of potential solutions and then convert it to its corresponding factor graph to apply the LBF, and find the most probable configuration of the distribution.

In this paper, we propose an algorithm based on Symmetric Non-negative Matrix Factorization (SNMF) (Kuang *et al*., 2012) to learn the structure of factor graph from a population of potential solutions. To the best of our knowledge, SNMF has never been used before for a learning factor graph structure or in any EDA algorithm. The factor graph clearly shows the variable correlations for ADFs.

We show the capability of our linkage learning approach for learning the factor graph for problems with overlapping factors, but as we are introducing a genetic algorithm capable of linkage learning (rather than an EDA), we limited this paper to investigate the FGGA only for optimizing the ADFs with non-overlapping linkage groups. This limitation arises as a result of applying genetic operators for searching for an optimum configuration of the variables in each building block. BB-wise genetic operators are shown to be ineffective in finding the optimum configuration of the variables in problems with overlapping building blocks (Yu, 2006).

Applying a sampling approach, like the one presented by Abbeel *et al*. (2006), for finding the best configuration of variables in problems with overlapping BBs using the learned factor graph seems to be straightforward, but since our concentration in this paper is on the linkage learning part of the algorithm, it is reserved for future work. In this paper, we assume variables of a factor node as BBs and we apply BB-wise crossover and BB-wise mutation to search for the optimum.

BB-wise mutation was first introduced by Sastry and Goldberg (2004) under the name of competent mutation. It is a local search in the BB neighborhood. This operator is investigated in the aforementioned paper and its effectiveness and scalability are shown. We employ the same mutation operator as proposed in (Sastry and Goldberg, 2004).

There are also other approaches that use recombination operators instead of sampling the distribution in the process of generating new potential solutions. Dependency Structure Matrix Genetic Algorithms (DSMGAs) (Yu and Goldberg, 2006), the Extended Compact Genetic Algorithm (ECGA) (Sastry and Goldberg, 2000) and Linkage Identification based on Non-linearity Check (LINC) (Munetomo and Goldberg, 1999) are some of these approaches. All of these and the FGGA can be categorized in the class of genetic algorithms capable of linkage learning.

In the proposed approach, we only need to calculate bivariate statistics. The input to our SNMF based algorithm to learn the factor graph structure is the matrix of pair-wise dependencies. There are other approaches that only use bivariate statistics to learn the model of the problem, which encode multivariate dependencies between variables of the problem. Gámez *et al*. (2008) introduce an approach to approximate multivariate dependency networks by using statistics of order two. The DSMGA (Yu and Goldberg, 2006) is another approach that only uses pair-wise dependencies to find the structure of the fitness function and solve the optimization problem. The original version of the Linkage Tree Genetic Algorithm (LTGA) (Thierens, 2010) uses multivariate statistics, but in the work of Pelikan *et al*. (2011), the algorithm is changed to use only bivariate statistics.

By applying the proposed algorithm, one can escape the computationally expensive task of learning the Bayesian network using the fit-to-data approaches and the costly task of computing multivariate statistics. We are proposing an approach to learn the factor graph, which is identical to additive decomposition using only bivariate statistics with less computational effort, in order to make the approach more feasible for the ultimate goal of optimizing real-world problem.

The paper is organized as follows. In the next section, background information on factor graphs and SNMF is presented. The relation between SNMF, the factor graph and linkage learning is explained in Section 3. In Section 4, the proposed algorithm is discussed in

detail. The algorithm is more thoroughly investigated in Section 5. In Section 6, the complexity of the algorithm is analyzed in terms of function evaluations as well as time and space requirements. Section 7 presents experimental results on some benchmark problems. Finally, the paper is concluded in Section 8.

2. Background

In this section, the background information on factor graphs and symmetric non-negative matrix factorization is reviewed.

2.1. Factor graph. A factor graph is a bipartite graph that encodes a distribution and expresses how variables of a problem factor into a product of local functions. Factor graphs subsume other graphical models such as Bayesian networks and Markov random fields. Many algorithms in various areas of artificial intelligence and machine learning use factor graphs and sum-product algorithms. But in most of them, the structure of the factor graph is supposed to be known in advance and the sum-product algorithm is used to compute marginal functions by distributed message-passing in the graph.

Suppose a function F of seven variables can be factored in three functions f_A, f_B and f_C:

$$\begin{aligned} F(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \\ = f_A(x_1, x_5, x_7) \times f_B(x_1, x_2, x_6) \\ \times f_C(x_2, x_3, x_4, x_7). \end{aligned} \tag{1}$$

The corresponding factor graph is shown in Fig. 1. There is a variable node (circular node) for each variable and there is a factor node (rectangular node) for each function. A variable node x_i is connected to the factor node f_I only if x_i is an argument of f_I.

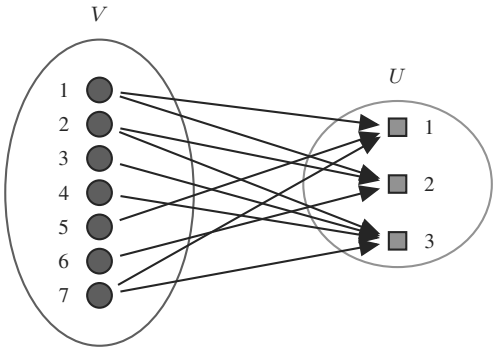

Fig. 1. Factor graph corresponding to Eqn. (1).

2.2. Symmetric non-negative matrix factorization. Given a nonnegative matrix X, Non-negative Matrix Factorization (NMF) consists in finding a lower-rank matrix approximation. The rank of a matrix is the number of linearly independent rows or columns of the matrix. So, NMF is formalized as

$$X \approx CG^T, \tag{2}$$

which can be achieved by calculating,

$$\min ||X - CG^T||, \tag{3}$$

where $||\cdot||$ denotes a distance metric like a divergence distance and $X \in \mathbb{R}_+^{m\times n}$, $C \in \mathbb{R}_+^{m\times k}$, $G \in \mathbb{R}_+^{n\times k}$.

NMF can be applied to a similarity matrix. In this case, matrix factorization is a symmetric non-negative matrix factorization problem with the formulation

$$\min_{H\geq 0} ||A - HH^T||_F^2, \tag{4}$$

where A is a similarity matrix of size $n \times n$ and H is a non-negative matrix of size $n \times m$. The largest entry in the i-th row of H indicates the assignment of the i-th data point due to the non-negativity of H.

Symmetric non-negative matrix factorization is developed specially for soft clustering in data sets with non-linear clusters. There are algorithms for symmetric non-negative matrix factorization which guarantee producing stationary point solutions. The uniqueness of SNMF in terms of factorization quality, the independence on the eigenspace of A, and the sensitivity to fluctuation in the similarity matrix A for a soft clustering application is shown by Kuang *et al*. (2012).

3. Symmetric non-negative matrix factorization, the factor graph and linkage learning

In SNMF, we are looking for a good factorization of a symmetric matrix, so that the lower rank of the matrix is approximated and the independent and dependent columns (or rows) are identified. In linkage learning, the goal is to find sets of dependent variables (linkage groups) as well as the optimum value for these variables knowing the dependency structure. The factor graph, on the other hand, is a graphical model which shows how variables of a problem factor into a product of local functions such that the variables of a function are dependent. So, using SNMF, we can learn linkage groups and utilize factor graphs to show the dependency structure.

If we assume that we have a symmetric probability affinity matrix $A \in \mathbb{R}^{n\times n}$ such that A_{ij} is a bivariate relationship between the variable x_i and x_j, and the bigger A_{ij} is, the more probable it is that x_i and x_j are from the same linkage group in the context of linkage learning (or the some cluster in the context of clustering, or the some factor in the context of the factor graph), then if we obtain

$A = HH^T$ we have factorized the bivariate relationship matrix into different groups (clusters or factors), which are independent of other groups (clusters or factors).

This can also be seen as a soft clustering approach. In soft clustering or fuzzy clustering each instance is associated with different clusters with different weights. This way, each instance can belong to more than one cluster. Here we would like to cluster the variables into different groups based on their dependencies.

The SNMF approach is depicted in Fig. 2. The first matrix represents the symmetric probability affinity matrix ($A_{n\times n}$) of a problem with $n = 10$, which is factorized into two matrices ($H_{n\times m}$ and $H^T_{m\times n}$). Darker shades correspond to bigger values. The factor graph is shown in Fig. 3. Each factor node corresponds to a linkage group (variables of a factor node form a linkage group).

4. Factor graph based genetic algorithm

In this section, the proposed algorithm which uses SNMF to learn a model of the data is introduced. The factor graph is used as the model to show the dependency structure. It is then employed to generate new potential solutions. The proposed algorithm is called the Factor Graph based Genetic Algorithm (FGGA). The FGGA uses bivariate statistics to learn multivariate dependencies between variables of the problem. The bivariate statistics that the FGGA uses is the mutual information between each pair of variables

$$\begin{aligned} \mathrm{MI}(X;Y) &= \sum_{x,y} P_{XY}(x,y) \log \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)} \\ &= E_{P_{XY}} \log \frac{P_{XY}}{P_X P_Y}. \end{aligned} \tag{5}$$

Mutual information is used as the bivariate statistics because it is one of the well-investigated metrics in the EDA community. Other pair-wise metrics can also be applied instead of mutual information.

After the calculation of MI for each pair of variables from the selected portion of the randomly generated population, the matrix of pair-wise dependencies E is constructed

$$E = \{e_{ij} = \mathrm{MI}(i;j)\}_{n\times n}. \tag{6}$$

We then factorize $E_{n\times n}$ into a bipartite graph using the SNMF approach. The resultant bipartite graph is a factor graph which represents the model of the problem and encodes multivariate dependencies between variables. The process of learning the structure of the factor graph will be explained in Section 4.1.

Having the factor graph and knowing the multivariate dependencies, new candidate solutions can be created. To sample those, one alternative is to use the Gibbs sampling approach or to apply the loopy belief propagation algorithm to find the most probable configuration. But as we are introducing a GA, we apply building block-wise operators to produce new potential solutions. Factor nodes of the factor graph contain the linkage group information. The set that contains all the neighbor nodes of each factor node is considered a linkage group.

BB-wise mutation and BB-wise crossover are used to sample new solutions. The former is used to make sure that the initial supply for the optimum configuration of each linkage group is adequate. The latter, on the other hand, is used to recombine different building blocks to get to the global optimum. These two operators are described in Section 4.2. The pseudo-code of the FGGA is depicted as Algorithm 1.

Algorithm 1. FGGA.

Output: Best solution found

```
Create random population.
repeat
    Evaluate the population.
    Select population by tournament selection.
    for N do
        for each variable i and variable j, do
            Compute e_ij = MI(i, j)
        end for
    end for
    Perform SNMF and find the factor nodes using Algorithm 2.
    Perform BB-wise mutation described in Section 4.2.1.
    Perform BB-wise crossover.
    Replace population.
until (maximum number of generations reached || population converged)
```

4.1. Factorization of the pair-wise dependencies matrix. In this section the factorization of the matrix E is explained. We first formulate the problem and then describe the learning approach.

4.1.1. Formulating the problem. Let $K(V,U,F)$ be a bipartite graph, where $V = \{v_i\}_{i=1}^n$ is a collection of variables and $U = \{u_p : p\}_{p=1}^m$ is a set of factor nodes (V and U are disjoint sets) while 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 pair-wise dependencies graph and find the corresponding bipartite graph, the similarity between v_i and v_j (e_{ij}) can be formulated as

$$\begin{aligned} e_{ij} &= \sum_{p=1}^{m} \frac{b_{ip}b_{jp}}{\lambda_p} = (B\Lambda^{-1}B^T)_{ij}, \\ \Lambda &= \mathrm{diag}(\lambda_1, \ldots, \lambda_m), \end{aligned} \tag{7}$$

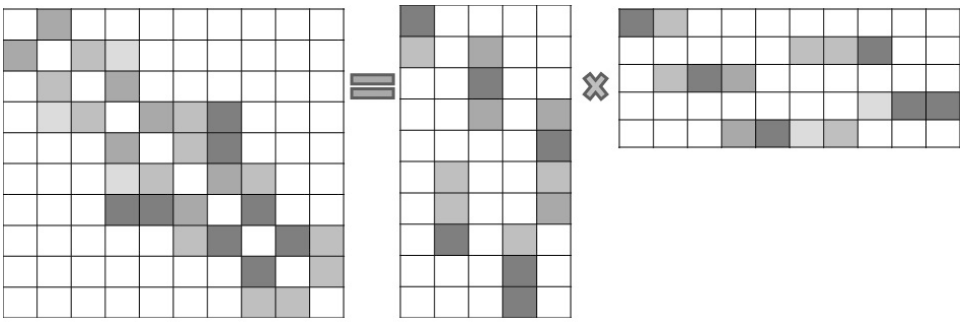

Fig. 2. Symmetric non-negative matrix factorization for a problem with 10 variables.

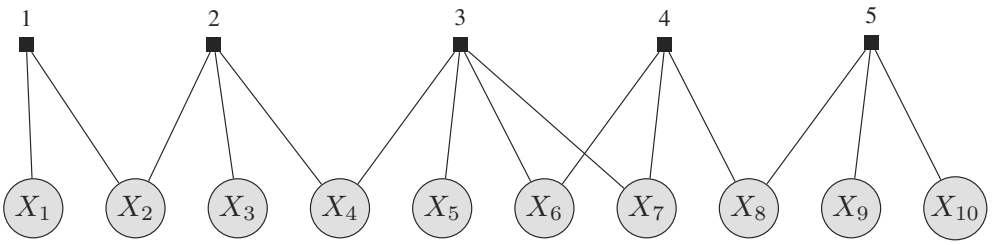

Fig. 3. Factor graph learned by SNMF for the above problem.

in the bipartite graph K (Zhou *et al.*, 2005), where $\lambda_p = \sum_{i=1}^{n} b_{ip}$ is the degree of vertex $u_p \in U$.

The above equation can be interpreted based on the random walks on graphs. Here 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

$$\begin{aligned} p(v_i, v_j) &= p(v_i)p(v_j|v_i) \\ &= d_i \sum_p p(u_p|v_i)p(v_j|u_p) \\ &= \sum_p \frac{p(v_i, u_p)p(u_p, v_j)}{\lambda_p}, \end{aligned} \tag{8}$$

where $d_i = p(v_i)$ is the degree of v_i and $p(v_j|v_i)$ is the conditional transition probability from v_i to v_j. With $b_{ip} = p(v_i, u_p)$, Eqns. (7) and (8) are the same. $p(u_p|v_i) = b_i p/d_i$ is the conditional probability of transitions from v_i to u_p and indicates how likely variable i belongs to factor nodes p.

The pair-wise dependency matrix and the corresponding factor graph are depicted in Figs. 2 and 3.

4.1.2. Learning the factorization. Based on Eqn. (7), the bipartite graph can be approximated by

$$\min \text{distance}(E, B\Lambda^{-1}B^T). \tag{9}$$

To make the problem easy, we use $H = B\Lambda^{-1/2}$. Then we have

$$\min_{H \in \mathbb{R}_+^{n \times m}} distance(E, HH^T) \quad \text{s.t. } h_{ip} \geq 0. \tag{10}$$

This problem is a symmetric non-negative matrix factorization (Lee and Seung, 2001). 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 (Eqn. (11)) between the two adjacency matrices,

$$\text{DD}(X, Y) = \sum_{ij} (x_{ij} \log \frac{x_{ij}}{y_{ij}} - x_{ij} + y_{ij}). \tag{11}$$

The following theorem (Yu *et al.*, 2006) is used as the optimization approach to find the minimum.

Theorem 1. (Yu *et al.*, 2006) *The distance is non-increasing under the following update rule:*

$$\hat{h_{ip}} = \frac{h_{ip}}{\sum_j h_{jp}} \sum_j \frac{e_{ij}}{(HH^T)_{ij}} h_{jp}. \tag{12}$$

The distance is invariant under the update rule if and only if H is at a stationary point of the distance.

After H is calculated, B can be calculated using $B = H\Lambda^{1/2}$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ where $\lambda_p = \sum_{i=1}^{n} h_{ip}$. The time complexity of Eqn. (12) is $O(m^2 \times n \times L)$, where L the number of nonzero entries in E. The proof of the convergence of the algorithm is given by Yu *et al*. (2006).

Based on the approach described above, we need to provide an upper bound on the number of factor nodes to the algorithm, which is not an uncommon task. We set this parameter to n in all the runs of the algorithm. In Section 5 it is shown that the performance of the algorithm does not depend on this parameter.

In order to provide an algorithmic scheme, we present the pseudo-code in Algorithm 2.

Algorithm 2. Learning the factor graph via SNMF.

Input: $E = \{e_{ij}\}_{n \times n}$. $\mathrm{maxFN} = n$

Output: Factor graph

1: Initialize: Start with an $n \times m$ matrix H, where $m = \mathrm{maxFN}$ and $H_{i \in V, j \in U} = [0, 1]$ is randomly set.
2: **repeat**
3: Update H based on Eqn. (12).
4: **until** H is invariant.
5: Construct $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ using $\lambda_p = \sum_{i=1}^{n} h_{ip}$.
6: Determine B using $B = H\Lambda^{1/2}$.

After performing the factorization we need to process the matrix B (or H). As explained before, the value $H_{i,j}$ indicates the weight with which the variable i is connected to the factor node j. We used a threshold, discarded all the elements of H less than 0.1 and constructed the factor graph. After this post-processing on matrix H, as we set $\mathrm{maxFN} = n$, we have n-factor nodes. In these n factor nodes, m-factor nodes are expected to have higher degrees and other factor nodes are probably duplicates of the other m high-degree factor nodes, or they are low-degree factor nodes and are discarded.

4.2. BB-wise operators. In this section, the genetic operators used for producing new potential solutions are described. Various forms of BB-wise genetic operators in different algorithms are used in the community. Here we used BB-wise mutation and BB-wise crossover presented by Sastry and Goldberg (2004).

4.2.1. BB-wise mutation. In the factorization phase, for each factor node p, its degree is computed as λ_p. We use the degree of each factor node as the certainty of the corresponding building block. This certainty metric shows the certainty of the algorithm about the correctness of the BB.

BB-wise mutation is only applied to the best individual of the population. Starting from the first non-mutated BB with a larger degree of certainty, all $2^{|BB|}$ individuals that can be created by $2^{|BB|}$ configurations of the BB are created and evaluated while the value of variables in the other BBs is kept constant. The best individual out of all of $2^{|BB|}$ individuals is retained in the population and the mutation process is continued on this individual until all the BBs are mutated. The same BB-wise mutation is used and investigated by Sastry and Goldberg (2004).

4.2.2. BB-wise crossover. Here we apply the uniform BB-wise crossover as it is used in many other approaches such as the ECGA and the DSMGA. In uniform BB-wise crossover, two parents are randomly selected from the mating pool and their building blocks in each partition are exchanged with the probability of 0.5.

5. Macroscopic analysis

In this section, we look closely at the SNMF approach by discussing the examples learned by this approach. In Section 5.1, we see examples of SNMF performance for learning linkage groups in decomposable problems in the context of the genetic algorithm and linkage learning. We analyze the matrix H with an example through different generations. In the second subsection, Algorithm 2 is used to find the linkage groups in problems with overlapping building blocks to see if the approach is capable of finding the building blocks for additively decomposable problems with overlapping BBs or not. The final subsection is dedicated to the analysis of the effect of the parameter maxFN used in Algorithm 2 on the performance of the algorithm.

5.1. SNMF for learning the linkage groups in problems with non-overlapping BBs. In Fig. 4, the $A_{n \times n}$ ($n = 50$) matrix of mutual information (or the MI matrix), the matrix $H_{n \times \mathrm{maxFN}}$ (SNMF resultant matrix, $\mathrm{maxFN} = 50$) and the linkage group matrix for the concatenated Trap-5 problem with 50 variables in different generations is depicted (a concatenated trap function with tight linkage groups is used for better visualization, so the algorithm is not sensitive to the location of variables in the string). The population size is set to 5600 and maxFN is set to n. The rows show the information of the generations 1, 3, 4 and 10 from top to bottom.

The MI matrix (the first column in the figure) is an $n \times n$ matrix containing the mutual information between each two variables. The intensity of each cell of the matrix is relative to its value. Lighter cells have larger values and darker cells smaller values. The matrix H (the second column in the figure) is an $n \times \mathrm{maxFN}$ matrix that shows the value with which each variable (rows of the matrix) is related to each factor node (columns of the matrix).

Like in the MI matrix, the intensity of each cell of the matrix is relative to its value. Lighter cells have larger values and darker cells smaller values. The linkage group matrix (third column in the figure) is an $n \times n$ matrix, and each of its entries corresponds to memberships in the linkage groups. If the value of the i-th row and the j-th column is one (white), the i-th and j-th variables are in the same linkage group, and if zero (black)—they are not. Since we are solving a concatenated Trap-5 problem, we expect the error-free MI matrix to have ten 5×5 white squares along the main diagonal, while all other cells of the matrix are expected to be zero (black). For the matrix H, in the error-free case, for each of the linkage groups there should be at least one factor node (corresponding to the columns in the matrix H) with a light bar, without any other disconnected light cells in the same column.

As is obvious in the figure, in the fourth generation (third row, third column) all the linkage groups are identified. But although the linkage groups are identified in the fourth generation, the optimum of the problem is not found up until the tenth generation (the forth row). This is due to the fact that some more generations are required after discovering the BBs, to search inside the BBs and mix the BBs in different individuals, in order to find an optimum solution. In this sample only BB-wise crossover is used to find the optimum configuration.

It is obvious from the figure that some of the columns (factor nodes) of the matrix H (the second column of Fig. 4) have very low degree (λ) (the degree of the factor nodes is defined in Section 4.1.1), so that their corresponding cells are almost zero (dark cells of the matrix). Along with the generations, the amount of noise in the MI and H matrices is getting smaller. For the first generation, the number of the factor nodes is set to $\mathrm{maxFN} = n = 50$. Along with the optimization algorithm, the number of unique and high-degree factor nodes approaches its actual value, which in this example is 10. The excess factor nodes (other 40 factor nodes) are either duplicates of the true factor nodes or have a very low degree. For example, in the bottom row of Fig. 4, 43 out of 50 columns have visually distinguishable degrees, and these 43 columns are duplicate and there are no false factor nodes among them. Ten unique factor nodes out of these 43 factor nodes are the actual factor nodes which are utilized as the input to other steps of the algorithm. Looking at the matrix H in the third row of the figure, it is obvious visually from the figure that there are almost no errors (disconnected light cells) at the columns with high degrees (columns with light bars).

As shown in the figure, although we do not provide the algorithm with the actual number of factor nodes of the problem, our SNMF approach can find the necessary number of factor nodes and thus the linkage groups. Initially, the number of factor nodes is set equal to that of variables of the problem. Again, it is shown that the optimization search for the optimum value which is performed by BB-wise crossover takes some more generations after completely learning the linkage groups (in the example, from generation 4 to generation 10 that the optimum is found). We may be able to decrease the necessary generations by applying some other sampling method, instead of the simple BB-wise crossover. More research on this issue is reserved for future work.

5.2. Parameter maxFN. In Fig. 5, the number of identified unique factor nodes during the algorithm for the Trap-5 problem with 50 variables is averaged over 100 runs and depicted as box plots. Here maxFN is set equal to the number of variables. During the algorithm, the number of identified unique factor nodes converges to the actual number of BBs (Fig. 5).

In Fig. 6, the total running time of the algorithm and the time spent on the factorization part of the algorithm for a different initial value of the maxFN parameter is depicted. As expected, the running time gets lower when maxFN is set closer to the real m.

5.3. SNMF for learning linkage groups in overlapping problems. In this subsection, the learned factor graphs for problems with overlapping building blocks are presented. In Fig. 7, the MI, H and linkage group matrices of a concatenated Trap-5 problem with 32 variables and 10 building blocks is presented for the population size 20000. Each two neighbouring BBs have two variables in common (see Fig. 8 for the structure of the problem). Here maxFN is set to $n/2$ in this example. As can be seen, the SNMF approach identified the linkage groups almost perfectly. As mentioned before, because BB-wise genetic operators are unable to find the optimum configuration for problems with overlapping BBs, we applied the SNMF learning approach only for

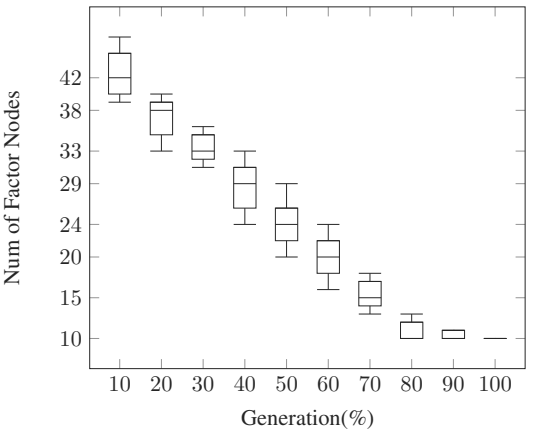

Fig. 5. Number of factor nodes for the concatenated Trap-5 problem with 50 variables over 100 runs.

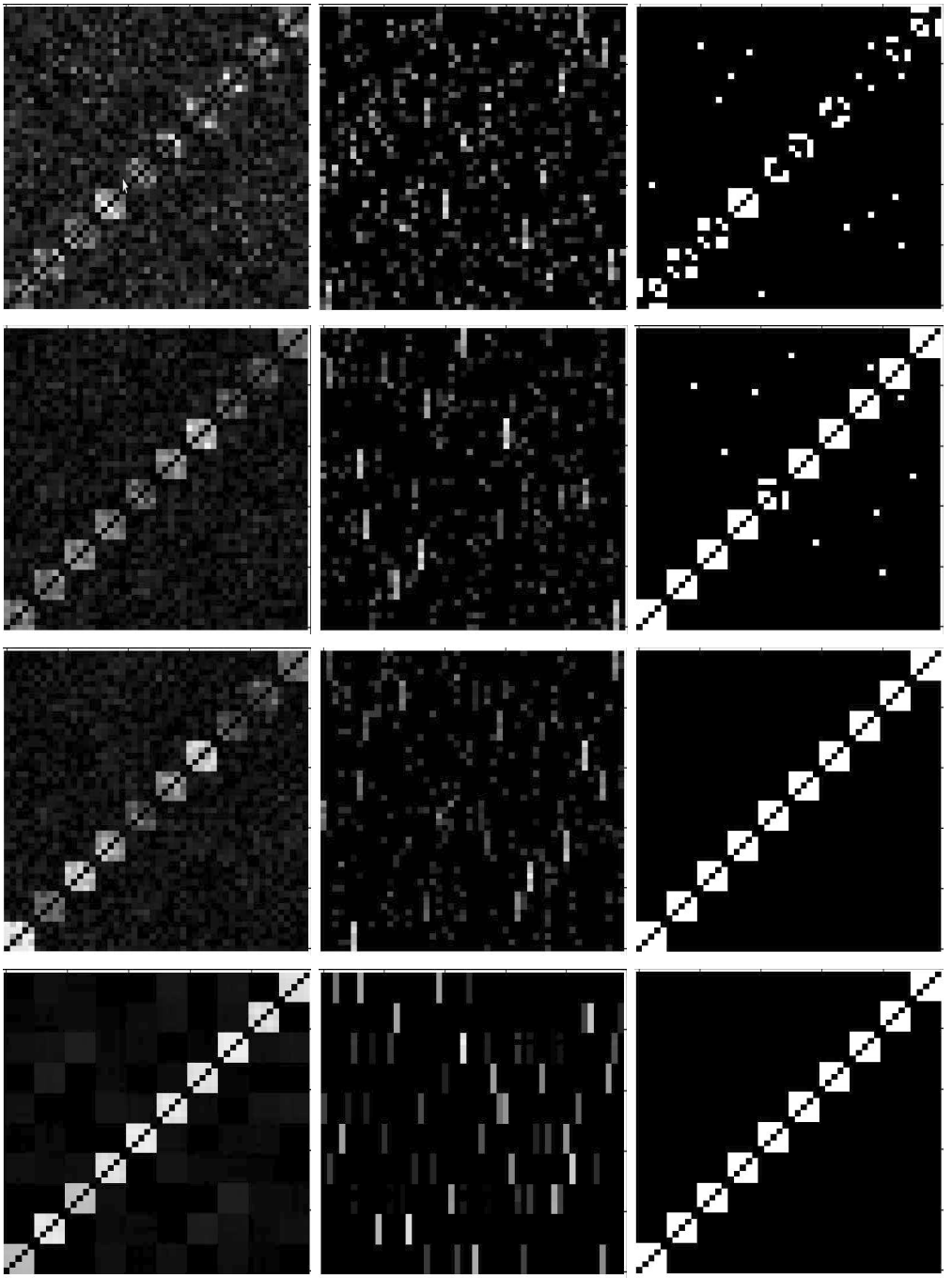

Fig. 4. Matrix of MI, SNMF and linkage groups (from left to right) in generations 1, 3, 4 and 10 (from top to bottom) for the Trap-5 problem with 50 variables.

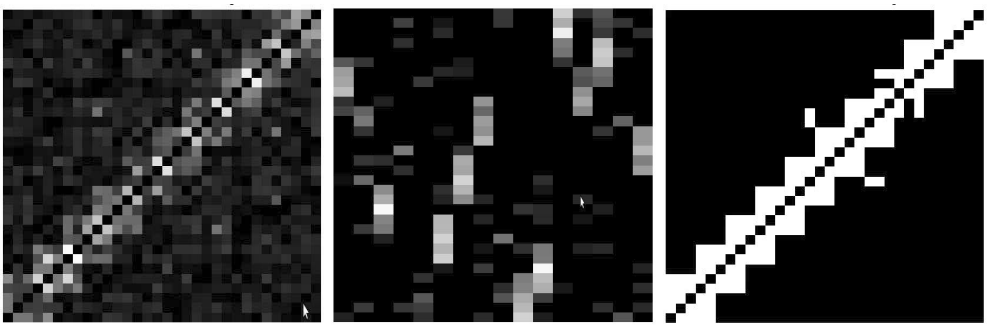

Fig. 7. 32×32 matrix of mutual information for the Trap 5 problem of size 32 with overlapping building blocks (2 variables overlap) (left), matrix H (middle), linkage groups constructed using SNMF (right).

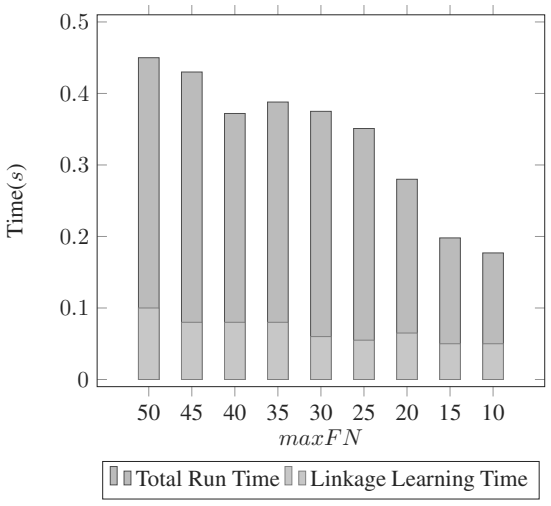

Fig. 6. Effect of maxFN on the total run time and linkage learning time for Trap 5 problems with 50 variables.

one generation for this problem.

6. Complexity analysis

In this section, the number of function evaluations and the time complexity of the FGGA algorithm are discussed. The algorithm is comprised of three main steps: (i) constructing the graph pair-wise dependencies, (ii) learning the factor graph, (iii) finding the optimum using BB-wise operators.

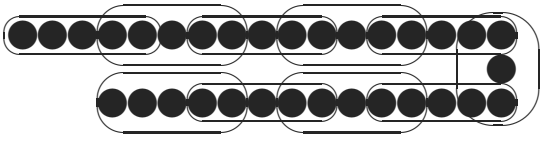

Fig. 8. Problem with 32 variables and 10 overlapping BBs.

6.1. Number of evaluations. In the first step of the algorithm, for constructing the pair-wise dependencies graph E, and in the third step for BB-wise operators, 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 the maximum length of k for each BB, at most $m \times 2^k$ evaluations are done during BB-wise mutation. Therefore, if the population size is N, the number of fitness evaluations in each iteration would be $m \times 2^k + N$, and if the number of generations is g, the overall number of evaluations of the algorithm would be $(m \times 2^k + N) \times g$. As shown by Yu *et al.* (2007), methods that use Shannons entropy to build models require a population of size $O(2^k n \log(n))$ to accurately build the model.

6.2. Time complexity. Pair-wise dependencies (mutual information) are computed by a pass through all the strings in the population. As MI values are determined for each pair of variables, 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 (Eqn. (11)), 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 Eqn. (12) 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.

7. Comparison results

In order to evaluate the proposed algorithm under different scenarios, we use different benchmark test functions, which will challenge the algorithm under different conditions.

7.1. Test functions.

7.1.1. OneMax problem. The OneMax problem is a simple problem consisting in maximizing the number of ones of a string $X = \{x_1, x_2, \ldots, x_n\}$ with $x_i \in \{0, 1\}$, so that $F(X) = \Sigma_{i=1}^{n} x_i$ is maximized. In Fig. 9, the population size required to solve this problem with the FGGA is depicted in a plot.

7.1.2. Trap function. An additively separable deceptive function of order k called k-deceptive is defined as the sum of single deceptive functions of order k (shown in Fig. 11) applied to non-overlapping k-bit partitions of solution strings.

$$f(x) = \begin{cases} \text{fHigh} & \text{if} \quad u(x) = k, \\ \frac{1}{k}\text{fLow} - u(x)\,\text{fLow} & \text{otherwise}, \end{cases} \tag{13}$$

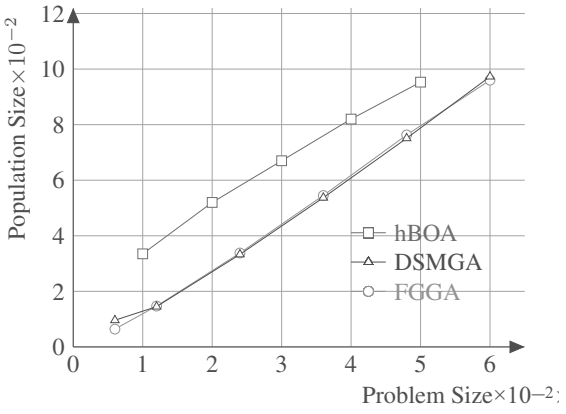

Fig. 9. Required population size to solve the OneMax problem with different sizes.

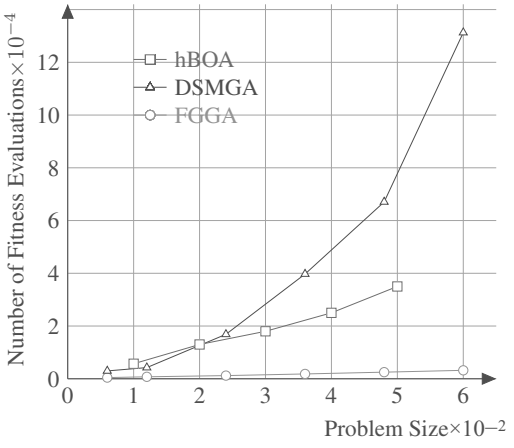

Fig. 10. Required number of fitness evaluations to solve the OneMax problem with different sizes.

where $u(x)$ returns the number of 1s in string x. Usually, $\text{fHigh} = 1$ and $\text{fLow} = 0.9$. It has one global optimum in individuals of all 1s and one local optimum in individuals of all 0s. The function is difficult because the local optimum has a larger basin of attraction.

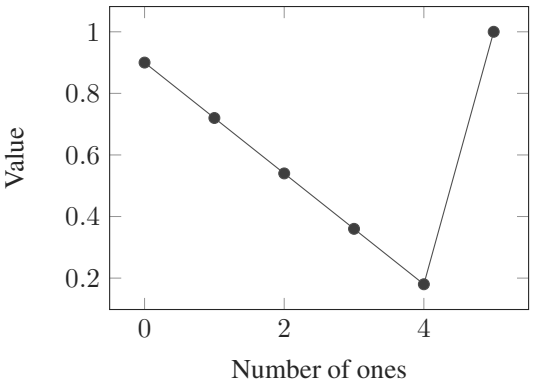

Fig. 11. Deceptive function for $k = 5$.

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.

7.1.3. Overlapping problems. As shown in Section 5, the SNMF approach is capable of identifying the linkage groups for overlapping problems, but BB-wise genetic operators are shown to be ineffective for finding the optimum values for overlapping BBs, although they have the linkage information. Therefore, the results for issues with overlapping problems are reserved for future work. These cases can be optimized effectively by using a probabilistic sampling approach.

7.2. Reference algorithms.

Among all the EDAs or probabilistic model building GAs, a few EDAs that use the factor graph as their model to represent and encode the multivariate dependencies (reviewed in Section 1) are not good candidates to be used as reference algorithms. For example, Santana *et al*. (2008) introduce an adaptive framework for EDAs, in which the different parts of the algorithm are not static and adapt based on different parameters. As our algorithm is not an adaptive EDA, the comparison of these two approaches would be between the apple and the orange. In the other approach, introduced by Mendiburu *et al.* (2007), a Bayesian network is learned from the population and then sampling is done using the corresponding factor graph and loopy belief propagation algorithm. As this work does not learn the factor graph structure, it is not used as the

reference algorithm either. Besides, the two mentioned approaches use computational costly methods of Gibbs sampling and the belief propagation algorithm, which makes them completely incomparable in complexity with the introduced approach, which is polynomial in terms of time and number of fitness evaluations.

Among all the EDA approaches, the hBOA (Pelikan, 2005) and the DSMGA (Yu and Goldberg, 2006) are selected as reference algorithms. The former is used because, first, it is a reputed and well-studied approach in the community and, second, it also learns a graphical model (Bayesian network) in order to represent the variable dependencies, so comparing the results of these two approaches may give the reader an evaluation on how these two graphical models work in contrast to each other.

The DSMGA is used as the second reference algorithm because, first, it uses a bivariate statistics to unveil the multivariate dependencies between variables of the problem and, second, it employs BB-wise operators to sample new potential solutions. In all of the runs of these algorithms, the parameters are set to their best configuration.

7.3. Results and discussion. The number of fitness evaluations for the hBOA, the DSMGA and the FGGA to solve concatenated 3-deceptive and 5-deceptive is depicted in Figs. 12 and 13. Comparing the results visually, the FGGA needs a smaller number of fitness evaluations and grows slower with regard to the problem size. As both the DSMGA and the FGGA use the same approach for searching for the optimum after finding the linkage groups, it seems that the linkage learning part of the two algorithms is the main reason for the difference in the required number of fitness evaluations. But this conclusion cannot be made about the hBOA, because here both the linkage learning and sampling are done differently. As can be seen, there is a difference in the growth of the number of fitness evaluations with the problem size between the hBOA and the other two approaches, since the hBOA uses multivariate interactions to build the model but the two other approaches use bivariate interactions. That difference in the scaling may be due to this difference in the model building process. Other problems, like NK-Landscapes or Ising spin glass cannot be efficiently optimized by the FGGA because here, BB-wise genetic operators are used to find the optimum. In order to solve these problems, the BB-wise genetic operators should be replaced by a sampling approach, which is reserved for future work.

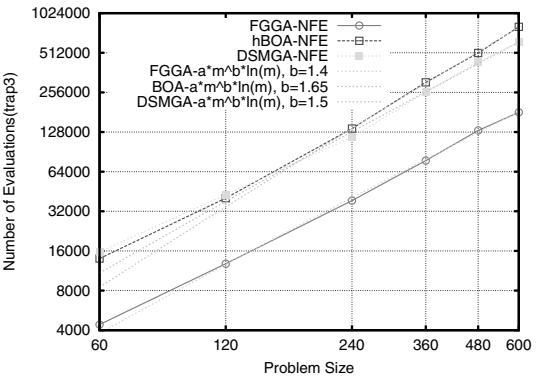

Fig. 12. Number of fitness evaluations for the hBOA, the DSMGA and the FGGA to solve concatenated 3-deceptive.

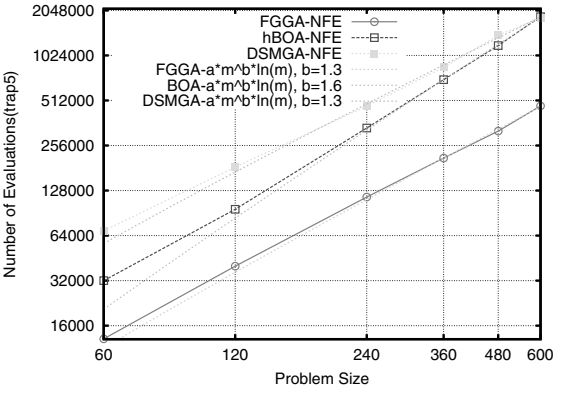

Fig. 13. Number of fitness evaluations for the hBOA, the DSMGA and the FGGA to solve concatenated 5-deceptive.

8. Conclusion

This paper introduces a factor graph based genetic 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. The FGGA 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 BB-wise mutation and BB-wise crossover. The proposed approach uses the well-studied mutual information as the measure of the pairwise dependencies between variables. To demonstrate the performance of the FGGA, the results on deceptive functions are reported. It is shown that the FGGA can solve all the tested problems with a polynomial number of fitness evaluations in polynomial time with respect to the length of the problem.

The results are compared with those of two well-known reference approaches, and it is shown that the performance of the three methods is comparable in

terms of the number of function evaluations. One of the advantages of the FGGA is that it learns the factor graph, which is naturally an ideal model to encode the variable dependencies in polynomial time. The algorithm description is easy and there is no black-box part there. 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 the FGGA has the potential of solving hierarchical problems by hierarchically factorizing the factor nodes. More analytical discussions on various problems and different parameters of the algorithm as well as experimental results on hierarchical problems are reserved for future works.

References

Abbeel, P., Koller, D. and Ng, A.Y. (2006). Learning factor graphs in polynomial time and sample complexity, *Journal of Machine Learning Research* **7**: 1743–1788.

Gámez, J.A., Mateo, J.L. and Puerta, J.M. (2008). Improved EDNA (estimation of dependency networks algorithm) using combining function with bivariate probability distributions, *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO'08, Atlanta, GA, USA*, pp. 407–414.

Kschischang, F.R., Frey, B.J. and Loeliger, H.A. (2001). Factor graphs and the sum-product algorithm, *IEEE Transactions on Information Theory* **47**(2): 498–519.

Kuang, D., Park, H. and Ding, C.H.Q. (2012). Symmetric nonnegative matrix factorization for graph clustering, *Proceedings of the 12 SIAM International Conference on Data Mining, Anaheim, CA, USA*, pp. 106–117.

Larrañaga, P. and Lozano, J.A. (2001). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Kluwer Academic Publishers, Norwell, MA.

Lee, D.D. and Seung, H.S. (2001). Algorithms for non-negative matrix factorization, *in* T.K. Leen, T.G. Dietterich and V. Tresp (Eds.), *Advances in Neural Information Processing Systems 13*, MIT Press, Cambridge, MA, pp. 556–562.

Mahnig, T. and Mühlenbein, H. (1999). FDA—a scalable evolutionary algorithm for the optimization of additively decomposed functions, *Evolutionary Computation,* **7**(4): 353–376.

Mendiburu, A., Santana, R. and Lozano, J.A. (2007). Introducing belief propagation in estimation of distribution algorithms: A parallel approach, *Technical Report EHU-KAT-IK-11-07*, University of the Basque Country, Bilbao.

Miquelez, T., Bengoetxea, E. and Larrañaga, P. (2004). Evolutionary computation based on Bayesian classifiers, *International Journal of Applied Mathematics and Computer Science* **14**(3): 335–349.

Mühlenbein, H. (2008). Convergence of estimation of distribution algorithms for finite samples, *Technical report*, Fraunhofer Institute for Autonomous Intelligent Systems, Sankt Augustin.

Mühlenbein, H. (2012). Convergence theorems of estimation of distribution algorithms, *in* S. Shakya and R. Santana (Eds.), *Markov Networks in Evolutionary Computation*, Adaptation, Learning, and Optimization, Vol. 14, Springer, Berlin/Heidelberg, pp. 91–108.

Mühlenbein, H. and Paaß, G. (1996). From recombination of genes to the estimation of distributions, I: Binary parameters, *in* H.M. Voigt, W. Ebeling, I. Rechenberger and H.P. Schwefel (Eds.), *Parallel Problem Solving from Nature IV*, Vol. 1141, Lecture Notes in Computer Science, Springer-Verlag, London, pp. 178–187.

Munetomo, M. and Goldberg, D.E. (1999). Identifying linkage groups by nonlinearity/nonmonotonicity detection, *Genetic and Evolutionary Computation Conference (GECCO-99), Orlando, FL, USA*, pp. 433–440.

Pelikan, M. (2005). *Hierarchical Bayesian Optimization Algorithm*, Springer-Verlag, Berlin/Heidelberg.

Pelikan, M., Goldberg, D.E. and Lobo, F.G. (2002). A survey of optimization by building and using probabilistic models, *Computational Optimization and Applications* **21**(1): 5–20.

Pelikan, M., Hauschild, M., and Thierens, D. (2011). Pairwise and problem-specific distance metrics in the linkage tree genetic algorithm, *MEDAL Report No. 2011001*, University of Missouri at St. Louis, St. Louis, MO.

Santana, R., Larraaga, P. and Lozano, J. (2008). Adaptive estimation of distribution algorithms, *in* C. Cotta, M. Sevaux and K. Srensen (Eds.), *Adaptive and Multilevel Metaheuristics*, Studies in Computational Intelligence, Vol. 136, Springer, Berlin/Heidelberg, pp. 177–197.

Sastry, K. and Goldberg, D. (2004). Designing competent mutation operators via probabilistic model building of neighborhoods, *in* K. Deb (Ed.), *Genetic and Evolutionary Computation, GECCO 2004*, Lecture Notes in Computer Science, Vol. 3103, Springer, Berlin/Heidelberg, pp. 114–125.

Sastry, K. and Goldberg, D. E. (2000). On extended compact genetic algorithm, *IlliGAL Report No. 2000026*, University of Illinois at Urbana–Champaign, Urbana, IL.

Thierens, D. (2010). Linkage tree genetic algorithm: First results, *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO'10), Portland, OR, USA*, pp. 1953–1958.

Yu, K., Yu, S. and Tresp, V. (2006). Soft clustering on graphs, *in* Y. Weiss, B. Schölkopf and J. Platt (Eds.), *Advances in Neural Information Processing Systems 18*, MIT Press, Cambridge, MA, pp. 1553–1560.

Yu, T.-L. (2006). *A Matrix Approach for Finding Extrema: Problems with Modularity, Hierarchy, and Overlap*, Ph.D. thesis, University of Illinois at Urbana–Champaign, Champaign, IL.

Yu, T.-L. and Goldberg, D.E. (2006). Conquering hierarchical difficulty by explicit chunking: Substructural chromosome compression, *8th Annual Conference on Genetic and Evolutionary Computation, (GECCO-2006), Seattle, WA, USA*, pp. 1385–1392.

Yu, T.-L., Sastry, K., Goldberg, D. E. and Pelikan, M. (2007). Population sizing for entropy-based model building in discrete estimation of distribution algorithms, *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07, London, UK*, pp. 601–608.

Zhou, D., Schölkopf, B. and Hofmann, T. (2005). Semi-supervised learning on directed graphs, *in* L.K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in Neural Information Processing Systems 17*, MIT Press, Cambridge, MA, pp. 1633–1640.

B. Hoda Helmi received her B.Sc. in computer engineering (software) from the Azad University of Mashhad at Mashhad, Iran, in 2004. She received her M.Sc. and Ph.D. in computer engineering (artificial intelligence) from the Iran University of Science and Technology at Tehran, Iran, in 2007 and 2014, respectively. She was a research scholar at the University of Missouri in St. Louis and a member of the Missouri Estimation of Distribution Algorithm Laboratory (MEDAL) in 2010–2012. This work was initiated while she was affiliated with the University of Missouri in St. Louis. Her research interests include genetic and evolutionary algorithms, probabilistic graphical models and discrete optimization.

Adel T. Rahmani received his B.Sc. degree in information systems from the Victoria University of Manchester, UK, in 1981, and the M.Sc. and Ph.D in computer engineering from Tokushima University, Japan, in 1993 and 1996, respectively. Since then he has been an assistant professor of artificial intelligence in the School of Computer Engineering at the Iran University of Science and Technology, Tehran, Iran. Currently, he is the head of the Artificial Intelligence Department in the School of Computer Engineering at the Iran University of Science and Technology. His research interests include genetic and evolutionary algorithms, estimation of distribution algorithms, multi-objective optimization, software engineering, formal methods and multi-agent systems.

Martin Pelikan received a Ph.D. in computer science from the University of Illinois at Urbana–Champaign in 2002. He is now a software engineer at Google. He was an associate professor at the Department of Mathematics and Computer Science at the University of Missouri in St. Louis up to 2012. This work was initiated while he was affiliated with the University of Missouri in St. Louis. In 2006, Pelikan founded the Missouri Estimation of Distribution Algorithms Laboratory (MEDAL). Prior to joining the University of Missouri, he worked at the Illinois Genetic Algorithms Laboratory (IlliGAL), the German National Center for Information Technology in Sankt Augustin, the Slovak University of Technology in Bratislava, and the Swiss Federal Institute of Technology (ETH) in Zurich. Pelikan has worked as a researcher in genetic and evolutionary computation since 1995. His most important contributions to genetic and evolutionary computation are the Bayesian Optimization Algorithm (BOA), the hierarchical BOA (hBOA), the scalability theory for the BOA and hBOA, and the efficiency enhancement techniques for the BOA and hBOA.

Received: 23 June 2013
Revised: 14 October 2013