AI Communications 24 (2011) 213–231
DOI 10.3233/AIC-2011-0498
IOS Press

Efficient model building in competent genetic algorithms using DSM clustering

Amin Nikanjam [*], Hadi Sharifi and Adel T. Rahmani
School of Computer Engineering, Iran University of Science and Technology, Tehran, 1684613114 Iran
E-mails: nikanjam@iust.ac.ir, hsharifi@comp.iust.ac.ir, rahmani@iust.ac.ir

Abstract. Detecting multivariate interactions between the variables of a problem is a challenge in traditional genetic algorithms (GAs). This issue has been addressed in the literature as the linkage learning problem. It is widely acknowledged that the success of GA in solving any problem depends on the proper detection of multivariate interactions in the problem. Different approaches have thus been proposed to detect and represent such interactions. Estimation of distribution algorithms (EDAs) are amongst these approaches that have been successfully applied to a wide range of hard optimization problems. They build a model of the problem to detect multivariate interactions, but the model building process is often computationally intensive. In this paper, we propose a new clustering algorithm that turns pair-wise interactions in a dependency structure matrix (DSM) into an interaction model efficiently. The model building process is carried out before the evolutionary algorithm to save computational burden. The accurate interaction model obtained in this way is then used to perform an effective recombination of building blocks (BBs) in the GA. We applied the proposed approach to solve exemplar hard optimization problems with different types of linkages to show the effectiveness and efficiency of the proposed approach. Theoretical analysis and experiments showed that the building of an accurate model requires $\mathrm{O}(n\log(n))$ number of fitness evaluations. The comparison of the proposed approach with some existing algorithms revealed that the efficiency of the model building process is enhanced significantly.

Keywords: Estimation of distribution algorithm, model building, linkage learning, dependency structure matrix

1. Introduction

Based on a well-known hypothesis [9,15], the implicit decomposition of a problem into sub-problems helps traditional genetic algorithms (GAs) to successfully solve the problem. This is achieved by the processing of the building blocks (BBs) of the problem that are groups of interacting genes, each of which constitute a partial solution to the problem. A group of highly linked locus or variables that forms a BB is called a linkage group. To solve the linkage learning problem, linkage groups must be detected first. Linkage groups form an interaction model of the problem that describes the interactions between variables of the problem. Once linkage groups are detected accurately, GA can perform the mixing task efficiently and accurately without BB disruption leading to appropriate convergence.

Several methods have been proposed in the literature for detecting linkage groups and providing proper BB mixing. These methods are categorized from different perspectives [3,21], all of them lie in one of the following two categories:

- Linkage identification methods.
- Estimation of distribution algorithms.

Tracing the fitness function changes in loci caused by perturbations is the technique used by the linkage identification methods to detect dependencies among loci. When linkage groups are detected, this information is used to find the optima of the problem. For example, the gene expression messy GA (GEMGA) [16] records fitness differences by perturbation of every locus for individuals in the population. Relations of genes are then detected according to the possibilities that the loci construct the local optima. The linkage identification by nonlinearity check (LINC) [22] identifies the linkage by detecting the nonlinearity of second order. It assumes that nonlinearity must exist within loci to form a BB. Offline utility of dependency structure matrix genetic algorithm (DSMGA) is another approach [40]. In this approach the DSM is employed to capture pair-wise dependencies of problems. Then a clustering algorithm is applied to the DSM to extract higher order

[*]Corresponding author. E-mail: nikanjam@iust.ac.ir.

interactions as linkage groups. Recently, an effective clustering algorithm has been proposed to detect linkage groups in a DSM [23]. However, because most of these methods need to evaluate fitness differences by pair-wise or more perturbations, they require O(n^2) or more fitness evaluations where n is the problem size [36].

EDAs [18] constitute a category of evolutionary algorithms that replace traditional genetic operators (crossover and mutation) with a usually probabilistic model of selected solutions that encodes the (in)dependencies between the variables of the problem. Machine learning techniques are employed to learn a model of (in)dependencies (interactions) from a population of selected solutions. This model is later used to generate new solutions that are incorporated into the population. In this way, this class of algorithms is able to overcome the linkage learning problem of traditional GAs.

EDAs may differ in the type of model they deploy to capture dependencies and the methods they use to learn and sample the model. Simple models like a probability vector in population-based incremental learning (PBIL) [30], compact genetic algorithm (cGA) [12] and univariate marginal distribution algorithm (UMDA) [20] are easy to learn and sample, but they are not capable to represent complex interactions. Although more complex models, like Bayesian networks in Bayesian optimization algorithm (BOA) [27] and marginal product model (MPM) in extended compact genetic algorithm (ECGA) [11], can be used to represent higher order interactions between variables, they mostly require a substantial amount of computational resources for learning [6,28].

Some researchers [26] have claimed that some successful EDAs have a polynomial rate of scale up with respect to the number of fitness evaluations, but they are computationally expensive yet. To solve large problems and/or problems with large BBs, it is necessary to enhance the efficiency of algorithm by auxiliary techniques like parallelization [2], hybridization [10], time continuation [33], evaluation relaxation [32] and incremental evolution [29].

Two computational bottlenecks for EDAs have been identified in the literature: *model building process* and *fitness evaluation* [14]. Building the model in particular is computationally complex and time consuming especially for problems with large BBs [6]. This subject is partially addressed through relaxation of the model building process [5] or by sporadic model building [28].

Several approaches have also been presented for efficient model building recently. Affinity propagation EDA [31] is based on learning a marginal product model by clustering a matrix of mutual information. Mutual information is learnt from the data using an efficient message-passing algorithm known as affinity propagation. This approach focuses on the balance between the accuracy of learning method and its time complexity, and shows that the application of more accurate learning algorithms does not necessarily yield better convergence results.

DSMGA is another new approach in model building [41,42] that extracts an interaction model from a DSM using clustering techniques. DSM, from the literature of project management and corporate organization [38], encodes the pair-wise dependencies of elements or variables in a complex system. GA then finds the optimum by using a BB-wise crossover that in turn uses BB information acquired by the clustering technique. DSMGA has been further extended with an explicit chunking method to solve hierarchical problems [41]. The interaction model used is explicit for problems with different structures. An offline usage of DSM analysis has been presented by Yu et al. [40] that thoroughly discuss the applicability, advantages and disadvantages of the offline model building.

Another new model building approach in EDAs is clustering over mutual information (ClusterMI) [4] that combines ideas from both ECGA and DSMGA. In this algorithm, the model building process of ECGA is replaced by a hierarchical clustering algorithm over mutual information. Mutual information of every pair of variables is calculated and stored in a matrix and a clustering method is then applied to determine the best partitioning of variables. Results show that ClusterMI has a more efficient model building process than ECGA, improving the overall running time of the algorithm. Building block identification by simultaneity matrix (BISM) [1] introduces direct identification of BBs from a simultaneity matrix. Each element of this matrix represents the dependency degree of two variables in the same way as mutual information matrix [4] and DSM. A partitioning algorithm is then applied to put the most interacting variables in the same subset. Comparison of BISM with BOA has revealed that although BOA needs fewer fitness evaluations, the computation of simultaneity matrix is faster than the construction of the Bayesian network in BOA.

D^5 [35,36] combines ideas from EDAs and perturbation-based methods to detect the linkages. In this approach individuals distribution is estimated using clustering them based on fitness values. Number of fitness evaluations needed by D^5 is less than other perturbation-based methods and it can solve problems that are difficult for EDAs more efficiently.

In the light of aforementioned related works, we aimed to propose a new comparatively much more efficient model building approach. In our approach, DSM is constructed based on the nonlinearity of fitness changes and it is then clustered offline by a new clustering algorithm to yield the interaction model. Clustered variables form an interaction model of the underlying problem that can be viewed as a decomposition of the problem. A GA with BB-wise crossover is then utilized to solve the optimization problem using the interaction model. It will be shown that constructing the DSM and finding an accurate interaction model is performed efficiently that reduces the computational cost of the model building process. Comparison to other well-known EDA, i.e. BOA, shows that the proposed approach can outperform BOA while solving testing problems.

The proposed approach can be considered as an offline model building approach [41] or a linkage detection method. Our approach is similar to the offline usage of DSMGA but a new effective DSM clustering is adopted that performs the clustering task without any additional information about the underlying problem. This paper is an extension of some previous works [23, 24]. The model building approach is discussed in more details, the accuracy of interaction model is investigated, complexity analysis is enriched, experiments are expanded and new results for overlapping problems are presented.

The rest of this paper is organized as follows. Section 2 presents an outline of our proposed approach. Section 3 provides a brief review of DSM and DSM clustering problem. Section 4 presents a detailed description of our proposed approach for model building including both the construction of DSM and DSM clustering algorithm. Section 5 analyzes the computational complexity of the proposed approach. Empirical results and quantitative comparison to other algorithms are presented in Section 6. Concluding remarks are discussed in Section 7.

2. Offline model building via DSM clustering

This section describes our approach to model building in EDAs based on DSM clustering. The basic idea is to use a clustering algorithm to transform pairwise dependencies, which are captured by DSM, into higher-order interactions as the interaction model. Although the offline utility of DSMGA [40] and offline DSM analysis in [41] uses a similar approach, we deploy a comparatively more efficient DSM clustering algorithm to derive an accurate interaction model. Furthermore, our approach lacks many disadvantages of existing related works and performs more appropriately.

Our approach has three main stages (Fig. 1). First, a DSM is created based on nonlinearity detection of the fitness function in a population of randomly generated individuals. DSM represents pair-wise dependencies between variables of the problem. The created DSM is then clustered to obtain the interaction model. Finally, the linkage groups information is fed into a BB-wise evolutionary algorithm to find the optima [35]. Both the DSM construction and the clustering of the constructed DSM are performed offline. Computational time saving, linkage information reusability, and population size estimation of GA are amongst the stated advantages of this approach [40,41]. The error in extract-

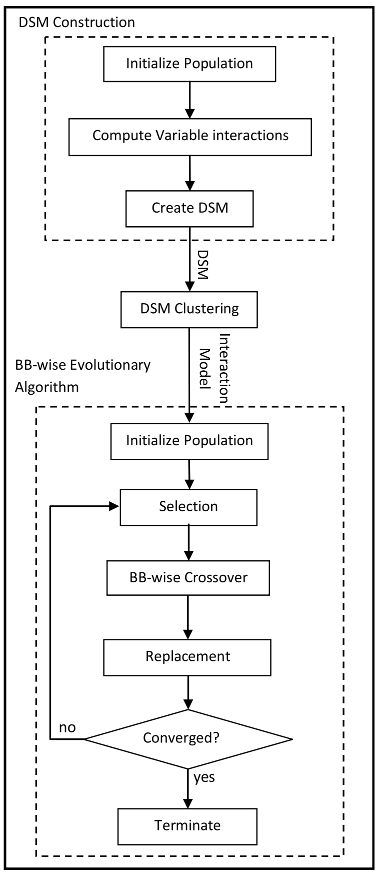

Fig. 1. Framework of the proposed approach.

ing linkage groups or quality of the retrieved interaction model was however an influential factor in the offline usage. Whilst the interaction model was extracted offline (out of main EAs loop) and BB-wise evolutionary algorithm worked based on this information (e.g., a GA with BB-wise crossover), non-detected linkages made the convergence difficult for the GA. It should be noted that BB-wise evolutionary algorithm only performed BB mixing without any effective tool for future linkage learning or adaptation. We analyzed our clustering algorithm to find the source of error and empirical results in all testing problems showed that by evaluating the fitness values of enough number of individuals, all the linkage groups can be found correctly. We pursued this strategy in the current proposed approach that is presented in this paper.

3. Dependency structure matrix and the clustering problem

3.1. Dependency structure matrix

A DSM by definition is an adjacency matrix of a graph where each entry d_{ij} represents the degree of dependency between node i and node j where each node stands for an element or variable in a complex system [38]. DSM entries d_{ij} can be real numbers or integers. The larger the d_{ij} is, the higher the dependency is between node i and node j. In this paper we focus on the binary domain, where $d_{ij} = 0$ means that there is no dependency between node i and node j, and $d_{ij} = 1$ means that node i and node j are completely dependent. The diagonal entries d_{ii} have no meaningful value and are usually set to zero, one or blacked-out. Fig. 2 illustrates an example of DSM.

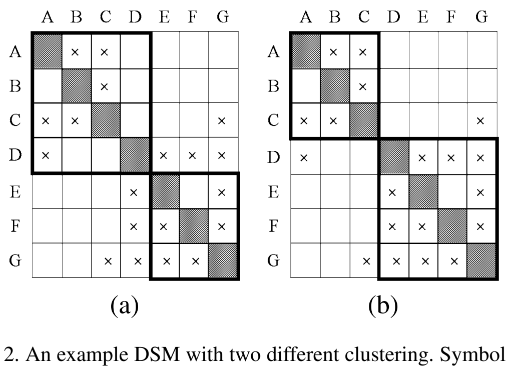

Fig. 2. An example DSM with two different clustering. Symbol '×' stands for the existence of a pair-wise dependency.

3.2. The problem of DSM clustering

The goal of DSM clustering is to find particular subsets of DSM elements (i.e., clusters) such that variables within a cluster are maximally interacting and variables within different clusters are minimally interacting. In other words, clustering of the DSM turns pair-wise interactions into higher-order interactions, namely linkage groups. Figure 2 illustrates two possible ways to cluster a DSM. Clusters obtained in Fig. 2(a) are {{A,B,C,D}, {E,F,G}} and clusters constructed in Fig. 2(b) are {{A,B,C}, {D,E,F,G}}. Black bordered squares represent clusters of the matrix.

DSM clustering is a complicated task that requires expertise. Usually a DSM can be clustered in different ways and determining the quality of clusters generated by each of them is not a simple and straightforward task [41]. Existing DSM clustering methods can be found in [34,37]. Their results show that the used objective function was incapable of accurately predicting good clustering because of the oversimplified objective functions [34]. To achieve better results, the minimum description length (MDL) metric has been employed to decide between good and bad clustering [41]. A simple GA was suggested then to find the best clusters using minimum description length criteria as the objective function. But in the reported experiments, a hill-climber is employed to reduce the computational time while finding the best clustering by the GA is computationally intensive. It should be noted that this clustering algorithm needs 'maximum number of clusters' as a crucial parameter to perform clustering well. This conflicts with the concept of black-box optimization and excludes the generality of utilization. In other words, it means that the maximum number of sub-problems must be determined before solving the problem.

4. Detection of linkage groups by DSM clustering

4.1. Constructing the DSM

DSM construction involves detecting pair-wise dependencies between the problem variables. Several methods are introduced in the literature for DSM construction [39]. One of these methods that is based on non-linearity detection [22] is presented in [42]. This method is based on investigating fitness change by injecting perturbations in a pair of variables. In this paper we use a similar approach that is briefly described here. The interaction between the ith variable and the

jth variable can be defined as:

$$|f_{a_i=0,a_j=1} - f_{a_i=0,a_j=0} - f_{a_i=1,a_j=1} + f_{a_i=1,a_j=0}|, \tag{1}$$

where $f_{a_i=x,a_j=y}$ is the fitness value of a schema where the ith variable is x, the jth variable is y and the rest are "don't care". However, computing the actual fitness value of schemata involves exploring the space of all possible combinations. A practical approach to this problem is approximating the value of $f_{a_i=x,a_j=y}$ by averaging the fitness value of some individuals in a population. Therefore, the information of interactions gathered from the population can be formulated as:

$$s_{ij} = |f'_{a_i=0,a_j=1} - f'_{a_i=0,a_j=0} - f'_{a_i=1,a_j=1} + f'_{a_i=1,a_j=0}|, \tag{2}$$

where $f'_{a_i=x,a_j=y}$ is the average value of $f_{a_i=x,a_j=y}$ based on a large enough population. The size of population that leads to an accurate approximation will be discussed in Section 5. Finally this information needs to be converted to a binary DSM using a threshold that is calculated by a two-means algorithm, a version of the k-means algorithm [13] with $k = 2$ [42]. The algorithm can be described as follows. First the DSM entries divided into two groups based on an initial threshold (e.g., the mean of the given data). Then, the average of mean of two groups calculated and the threshold is updated to this value. Dividing entries into two groups is repeated using the updated threshold. Updating the threshold and dividing are repeated until the two groups do not change anymore. The last used threshold will be considered as the final desired threshold.

In the proposed method no selection method is adopted in this stage and all randomly generated individuals in the population are used to construct the DSM. An algorithmic description of DSM construction is presented in Fig. 3. Figure 4 illustrates the constructed DSM for a $m - k$ trap function [8] with $m = 3$ and $k = 5$ by evaluating the fitness value of 300 individuals.

```
Create random population.
Evaluate the population.
for each variable $i$ and variable $j$,
  for each individual $a$ in the population,
  update $f'_{a_i=x,a_j=y}$

    $s_{ij} = |f'_{a_i=0,a_j=1} - f'_{a_i=0,a_j=0} - f'_{a_i=1,a_j=1} + f'_{a_i=1,a_j=0}|$

newThreshold = mean of $s_{ij}$ values
Do {
  for each variable $i$ and variable $j$,
    if ($s_{ij}$ > newThreshold)
        $DSM[i, j] = 1$
    else
        $DSM[i, j] = 0$
  oldThreshold = newThreshold
  newThreshold = mean of two means of binary DSM
} while (newThreshold ≠ oldThreshold)
```

Fig. 3. Algorithmic description of DSM construction.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1		1	1							1		1	
2	1	1		1	1	1			1					1	1
3			1	1	1										
4	1	1	1	1	1				1				1		
5	1	1	1	1	1				1		1				
6		1				1	1		1		1			1	
7						1	1	1	1	1					
8							1	1	1	1					
9		1		1	1	1	1	1	1	1			1	1	
10							1	1	1	1					
11					1	1					1		1	1	1
12	1											1	1	1	1
13				1					1		1	1	1	1	1
14	1	1				1			1		1	1	1	1	1
15		1									1	1	1	1	1

Fig. 4. An example of a DSM constructed for a $m - k$ function with $k = 5$ and size of 15. Density of the group {6,7,8,9,10} (black bordered) is $21/25 = 0.84$, so this group is considered as an initial cluster.

4.2. Clustering the DSM

In this subsection a new DSM clustering algorithm is introduced that does not need crucial parameters like maximum number of clusters, which are necessary in other clustering methods [41]. There are some parameters that must be initialized for proper performance of the algorithm but some of them are computed automatically and others are not sensitive to the underlying problem.

The proposed clustering algorithm is based on natural groups of dependent variables that were seen in the DSM and are comprised of several phases to generate and revise these natural groups. The first phase starts with grouping the variables (namely *initial clusters*) that are dependent on each other based on the density of groups in the constructed DSM. The second phase involves constructing *secondary clusters* using initial clusters by moving variables between clusters based on co-occurrence of variables in the clusters. The third and the fourth phase of the clustering algorithm consist of revising secondary clusters based on DSM entries to obtain the final clusters. Figure 5 illustrates the flowchart of the proposed clustering algorithm.

The first phase begins by grouping the variables that are dependent on each other according to the DSM constructed previously. There would be a group for each variable. From these initial groups of variables,

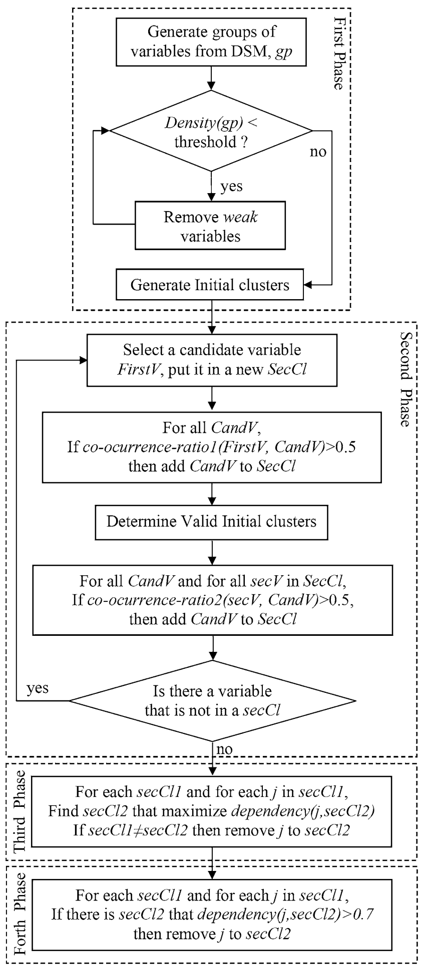
First Phase
Generate groups of variables from DSM, gp
Density(gp) < threshold ?
no
yes
Remove weak variables
Generate Initial clusters
Second Phase
Select a candidate variable FirstV, put it in a new SecCl
For all CandV, If co-ocurrence-ratio1(FirstV, CandV)>0.5 then add CandV to SecCl
Determine Valid Initial clusters
For all CandV and for all secV in SecCl, If co-ocurrence-ratio2(secV, CandV)>0.5, then add CandV to SecCl
Is there a variable that is not in a secCl
yes
no
Third Phase
For each secCl1 and for each j in secCl1, Find secCl2 that maximize dependency(j,secCl2) If secCl1≠secCl2 then remove j to secCl2
Forth Phase
For each secCl1 and for each j in secCl1, If there is secCl2 that dependency(j,secCl2)>0.7 then remove j to secCl2

Fig. 5. Flowchart of the proposed clustering algorithm.

the initial clusters are built using a density threshold and a heuristic for eliminating the variables, called *weak variables*, from the initial groups. Constructing the initial clusters from initial groups of variables is done as follows. First, the density of each group of variables is calculated using Eq. (3) and it is compared with the density threshold

$$Density(G) = \frac{\sum_{i,j \in G} DSM_{i,j}}{|G|^2}, \tag{3}$$

where G is a group of variables, $|G|$ is the number of variables in G and DSM is the dependency structure matrix that is constructed as described in the previous subsection. The density will take its maximum value of 1 when all the variables in G are dependent based on DSM entries (value of 1 for all entries in the matrix). On the other extreme, when all variables in G are independent, the density will be zero.

If the density of a cluster is more than the threshold, that group is identified as an initial cluster, *initCl*. Otherwise, the group is pruned by a heuristic that eliminates *weak variables* to become an initial cluster. A variable in a group is called weak when it depends on less than half of all the variables in the group (dependency of the variables is determined based on DSM). A weak variable is eliminated from the group and the density of the group is again calculated to see if it is an initial cluster or not. The density threshold is determined experimentally and it is set to 0.75. The first phase terminates when no new initial cluster is generated by removing weak variables.

For better understanding, a trace of the execution of the proposed algorithm is presented step by step. An example of initial groups in a DSM is presented in Fig. 4. Figure 6 illustrates all initial clusters that are generated during the execution of the first phase of the algorithm on the DSM in Fig. 4. Figure 7 shows an algorithmic description of the first phase of the proposed DSM clustering algorithm.

The second phase involves constructing secondary clusters. This phase has a main loop that itself consists of three steps. In the first step, a *secondary cluster* is generated. In the second step, *valid initial clusters* are determined and in the third step, the secondary cluster generated in the first step is expanded based on valid initial clusters. The main loop repeats until each variable is a member of a secondary cluster.

Step 1. In the first step of the loop, an arbitrary variable, not being clustered into any secondary cluster yet, is selected to begin with. The selected variable called

Phase 1: initial clusters	Phase 2: secondary clusters	Phase 3: final clusters
{**1**,2,4,5,14}	{1,2,4,5,9,14}	{1,2,3,4,5}
{1,**2**,4,5,6,9,14}	{3}	{6,7,8,9,10}
{**3**,4,5}	{6}	{11,12,13,14,15}
{1,2,**4**,5,9}	{7,8,10}	
{1,2,4,**5**,9}	{11,12,13,15}	
{2,**6**,9,11,14}		
{6,**7**,8,9,10}		
{7,**8**,9,10}		
{6,7,8,**9**,10}		
{7,8,9,**10**}		
{6,**11**,13,14,15}		
{1,**12**,13,14,15}		
{11,12,**13**,14,15}		
{2,6,9,11,13,**14**,15}		
{11,12,13,14,**15**}		

Fig. 6. Trace of the proposed clustering algorithm on the DSM illustrated in Fig. 4.

```
for each variable $i$ in the variable set,
  create a new group, $gp_i$ containing variable $i$.
    for each variable $j$,
      if ($DSM[i,j]==1$),
        add $j$ to $gp_i$.
for each $gp_i$,
    while(($Density(gp_i) < densityThreshold$) or
    (there is a weak variable))
        remove the variable with minimum
        dependency on other variables in $gp_i$
    initCl = $gp_i$
```

Fig. 7. Algorithmic description of DSM clustering in the first phase.

firstV constructs a new secondary cluster *secCl* with one member *firstV*. The remaining variables that do not belong to any secondary clusters are called *candidate variables*, *candV*. A metric named *co-occurrence-ratio1* is calculated that shows the co-occurrence ratio (simultaneous occurrence) of *firstV* and a *candV*. This is an extension of the idea of 'tightness detection' [21] which asserts that if two variables are highly linked, it is expected to observe them simultaneously in other linkage groups (clusters). Based on this ratio, it is decided whether or not the *candV* should be included in *secCl* or not. The *co-occurrence-ratio1* is the number of initial clusters that contain both the candidate variable *candV* and the first variable *firstV* divided by the number of initial clusters that contain either *firstV* or *candV* or both. This is formulated in

$$co-occurrence-ratio1(candV,firstV) = \frac{|\{initCl|candV \in initCl\ and\ firstV \in initCl\}|}{|\{initCl|candV \in initCl\ or\ firstV \in initCl\}|}, \quad (4)$$

where *initCl* is an initial cluster. If the *co-occurrence-ratio1* is more than 0.5, the candidate variable *candV* is added to the secondary cluster *secCl*. After calculating the *co-occurrence-ratio1* between *firstV* and all of the candidate variables, step 2 is terminated.

Step 2. Now, based on *secCl*, a new term is defined: those initial clusters that contain at least half of the variables in *secCl* are considered to be *valid initial clusters*, i.e., *valCl*.

Step 3. To complete the construction of the *secCl*, for each remaining candidate variables, a similar metric to *co-occurrence-ratio1* is calculated. The *co-occurrence-ratio2* evaluates the co-occurrence of each variable *secV* in the *secCl* and a *candV* based on valid initial clusters. If the *co-occurrence-ratio2* of a candidate variable and one of variables in *secCl* is more than 0.5, that candidate variable is added to *secCl*. This metric is calculated for every candidate variable and every variable in the secondary cluster (except the *firstV*). This process continues until no new candidate variable is added to *secCl*. The *co-occurrence-ratio2* is the number of valid initial clusters containing both the candidate variable *candV* and a variable in *secCl*, divided by the number of valid initial clusters that contain either *candV* or the variable in *secCl* or both. This is formulated in

$$co-occurrence-ratio2(candV,secV) = \frac{|\{valCl|candV \in valCl\ and\ secV \in valCl\}|}{|\{valCl|candV \in valCl\ or\ secV \in valCl\}|}, \quad (5)$$

where *valCl* is a valid initial cluster and *secV* is a variable in *secCl* except the *firstV*.

After accomplishing these three steps, the first secondary cluster is constructed. If there is any variable that is not in a secondary cluster, the loop is repeated and new secondary clusters are generated through steps 1–3. The second phase terminates when there is no variable that does not belong to a secondary cluster. Algorithmic description of the second phase is presented in Fig. 8. While Fig. 6 presents the obtained secondary clusters for the DSM in Fig. 4, a trace of steps 1–3 is shown in Figs 9 and 10.

The third phase of clustering consists of revising secondary clusters based on DSM entries to obtain final clusters. In this phase, for each variable in each secondary cluster, the relative dependency to other secondary clusters is calculated. The relative dependency of a variable to a secondary cluster is the number of

```
for each variable $i$ that is not in a secondary cluster,
  add $i$ to new empty secondary cluster, *secCl*.

  Step 1: Constructing secondary cluster
  for each variable $j$ such that $i \neq j$ and $j$ is not in a secondary cluster,
        if (*co-occurrence-ratio1(j, i)* $\geqslant 0.5$)
            add $j$ to *secCl*.
  Step 2: Determining valid initial clusters
  for each initial clusters *initCl*,
        if (*initCl* contains at least half of variables of *secCl*)
            add *initCl* to the set of valid initial clusters.
  Step 3: Revising secondary clusters
  for each variable $j$ such that is not in *secCl*,
      for each variable $k$ that is in *secCl* and $k \neq i$,
        if (*co-occurrence-ratio2(j,k)* $\geqslant 0.5$)
            add $j$ to *secCl*.
```

Fig. 8. Algorithmic description of DSM clustering in the second phase.

Initial clusters	Secondary clusters	Calculations
{1,2,4,5,14}	{1,2}	Co-occurrence-ratio1(2,1) =
{1,2,4,5,6,9,14}		$4/7 \geqslant 0.5$
{3,4,5}	{1,2}	Co-occurrence-ratio1(3,1) =
{1,2,4,5,9}		$0/6 \leqslant 0.5$
{1,2,4,5,9}	{1,2,4}	Co-occurrence-ratio1(4,1) =
{2,6,9,11,14}		$4/6 \geqslant 0.5$
{6,7,8,9,10}	{1,2,4,5}	Co-occurrence-ratio1(5,1) =
{7,8,9,10}		$4/6 \geqslant 0.5$
{6,7,8,9,10}	{1,2,4,5}	Co-occurrence-ratio1(6,1) =
{7,8,9,10}		$1/10 \leqslant 0.5$
{6,11,13,14,15}	{1,2,4,5}	Co-occurrence-ratio1(7,1) =
{1,12,13,14,15}	.	$0/9 \leqslant 0.5$
{11,12,13,14,15}	.	.
{2,6,9,11,13,14,15}	.	.
{11,12,13,14,15}	{1,2,4,5}	Co-occurrence-ratio1(15,1) =
		$1/9 \leqslant 0.5$

Fig. 9. Trace of step 1 in the second phase of the proposed algorithm, constructing secondary clusters.

Valid initial clusters for the secondary cluster {1,2,4,5}	Secondary cluster	Calculations
{1,2,4,5,14}	{1,2,4,5,9}	Co-occurrence-ratio2(9,2) =
{1,2,4,5,6,9,14}		$3/4 \geqslant 0.5$
{3,4,5}	{1,2,4,5,9,14}	Co-occurrence-ratio2(14,1) =
{1,2,4,5,9}		$2/4 \geqslant 0.5$
{1,2,4,5,9}	{1,2,4,5,9,14}	Co-occurrence-ratio2(6,1) =
		$1/4 \leqslant 0.5$
	{1,2,4,5,9,14}	Co-occurrence-ratio1(6,2) =
	.	$1/6 \leqslant 0.5$
	.	
	{1,2,4,5,9,14}	

Fig. 10. Trace of steps 2 and 3 in the second phase of the proposed algorithm, constructing secondary clusters.

variables in the secondary cluster that are dependent on that variable (according to DSM) to the total number of variables in the secondary cluster.

$$\mathit{Dependency}(\mathit{secV}, \mathit{secCl}) = \frac{\sum_{y \in \mathit{secCl}} \mathit{DSM}_{\mathit{secV},y}}{|\mathit{secCl}|}, \tag{6}$$

where |*secCl*| is the number of variables in *secCl*. Assume a variable *secV* that is in a secondary cluster *secCl1*. If dependency of *secV* and any other secondary cluster (for example *secCl2*) is more than the dependency of *secV* to *secCl1*, then *secV* should be removed from *secCl1* and added to *secCl2*. To avoid preference of smaller clusters to bigger ones, when dependency of a variable to a cluster is equal to 1, it is changed to $1 - (1/(|\mathit{secCl2}| * 2))$. For example, when dependency of a variable to a cluster with two members is $2/2$ and its dependency to a cluster with 10 members is $8/10$, the cluster with 10 members is considered to be the most dependent cluster. The third phase will terminate when no variable changes its cluster; this is shown in Fig. 11 in an algorithmic form. Figure 12 shows the trace of the third phase.

The fourth phase is very similar to the third phase and is performed to obtain refined final clusters especially for overlapping problems. In the overlapping problems, some variables are the same in different clusters. Observations revealed that all overlapping variables are not detected after the third phase. Therefore, similar approach to the third phase is applied

```
Do{
  for each secondary cluster *secCl1*,
    for each variable $j$ such that $j$ belong to *secCl1*,
      for each secondary cluster *secCl2*,
          calculate $\mathit{Dependency}(j, \mathit{secCl2})$.
          if ($\mathit{Dependency}(j, \mathit{secCl2}) = 1$)
              $\mathit{Dependency}(j, \mathit{secCl2}) =$
              $1 - 1(\mathit{secCl2} * 2)$.
              find secondary cluster *secCl2* that has maximum
              $\mathit{Dependency}(j, \mathit{secCl2})$
          if ($\mathit{secCl1} \neq \mathit{secCl2}$)
              remove $j$ from *secCl1* and add it to *secCl2*.
} while (no change in clusters)
```

Fig. 11. Algorithmic description of DSM clustering in the third phase.

Secondary clusters	Variables in *secCl1*	Calculations
secCl1 = {1,2,4,5,9,14}	1	*Dependency*(1,*secCl1*) = 5/6 and is maximum
secCl2 = {3}		
secCl3 = {6}	2	*Dependency*(2,*secCl1*) = 6/6 and is maximum
secCl4 = {7,8,10}	.	
secCl5 = {11,12,13,15}	.	
	.	
	9	*Dependency*(9,*secCl1*) = 5/6 but *Dependency*(9,*secCl4*) = 3/3, so 9 is moved to *secCl4*
	14	*Dependency*(14,*secCl1*) = 3/5 but *Dependency*(14,*secCl5*) = 4/4, so 14 is moved to *secCl5*

Fig. 12. Trace of the third phase, revising secondary clusters.

```
Do{
threshold = $0.7$
for each secondary cluster secCl1,
    for each variable $j$ such that $j$ belong to secCl1,
        for each secondary cluster secCl2,
            calculate Dependency($j$, secCl2).
            if (Dependency($j$, secCl2) = 1)
                Dependency($j$, secCl2) =
                $1 - 1/(|secCl2| * 2)$.
            if (Dependency($j$, secCl2) > threshold)
                add $j$ to secCl2.
} while (no change in clusters)
```

Fig. 13. Algorithmic description of DSM clustering in the fourth phase.

in the fourth phase with a fixed threshold of dependency to extract all of such variables. In the case non-overlapping problems, this procedure has not an influential impact on the final clusters. This phase is shown in Fig. 13.

5. Complexity analysis

In this section, the complexity analysis of the proposed linkage group detection algorithm is presented. Since this algorithm is comprised of two main parts, namely of constructing and clustering DSM, the computational complexity of them are analyzed separately in this section with respect to memory and time.

5.1. Memory complexity

In the process of DSM construction, the main portion of the memory used, is the memory consumed for keeping the population of individuals. Based on this population, pair-wise dependencies are calculated. But keeping all the individuals in the memory is not necessary and calculating s_{ij} values may be done incrementally, reducing the required population size to O(1). Therefore, the total number of individuals evaluated by the fitness function to construct the DSM is more plausible than the size of population. Besides the memory used for keeping the population of individuals, four $n \times n$ matrix (O(n^2)) are needed to store pair-wise dependencies, where n is the problem size.

The amount of memory required by the proposed DSM clustering is O(n^2). In the first phase, at most n initial clusters may be created, each of which may have (at most) n members. Therefore, it requires an $n \times n$ matrix to store the initial clusters. In the second phase, secondary clusters are constructed that require exactly an array with the length of n. The third and the fourth phase do not require any extra memory. So overall, the proposed DSM clustering memory requirement is bounded by O(n^2).

5.2. Time complexity

There are two ways to report the time complexity of DSM construction. The first one is reporting the number of fitness evaluations based on the justification that in complex real-world applications, fitness evaluation is computationally expensive. Therefore, the time required to evaluate a candidate solution dominates the time spent for other parts of the algorithm [26]. The other one is counting the number of operations in the algorithm. Both of these methods are presented here.

During DSM construction, in a non-incremental manner, if the population size is N, the number of fitness evaluations would be N. It has been shown [22] that for $m - k$ trap problem wherein the order of each m trap function is bounded by k (bounded difficulty problems), the number of individuals that must be evaluated by the fitness function to detect all the pair-wise non-linearities is bounded by O($n^2 \times 2^k$). By considering a fixed error rate in detecting pair-wise linkages [39,43], N can be derived by the following derivations.

To construct the DSM, the value of s_{ij} is evaluated for every pair of variables by Eq. (2) based on a population of N individuals. In the $m - k$ trap problem, if we assume the ith and jth variables to be dependent, similar to [43], the distribution of s_{ij} can be approximated as a Gaussian distribution when N is large [7] with the following mean and variance:

$$E_{\text{dep}}[s_{ij}] = \frac{c_1 d}{2^{k-2}}, \tag{7}$$

$$\text{Var}_{\text{dep}}[s_{ij}] = \frac{c_2 m \sigma_{\text{BB}}^2}{N}, \tag{8}$$

where d is the minimal fitness difference between competing BBs, m is the number of BBs, k is the order of each BB, σ_{BB}^2 is the fitness variance of a BB and c_i is a constant. If two variables are independent, $E_{\text{indep}}[s_{ij}]$ is negligible for large N and $\text{Var}_{\text{indep}}[s_{ij}]$ tends to zero. Some experiments are performed to verify these derivations using $m-k$ trap with $m=9$ and $k=5$. Results are averaged over 5000 independent runs. As it can be seen in Fig. 14, $E_{\text{dep}}[s_{ij}] - E_{\text{indep}}[s_{ij}]$ is independent of N for large N. Another important observation is about $\text{Var}_{\text{dep}}[s_{ij}]$ and $\text{Var}_{\text{indep}}[s_{ij}]$ that are $O(N^{-1})$ when N is large.

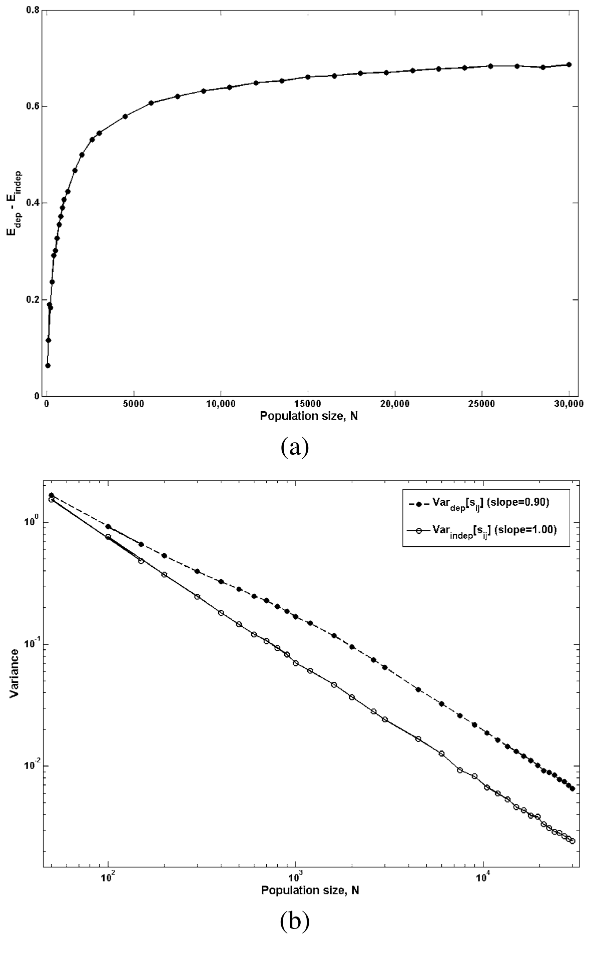

Fig. 14. The effect of population size on (a) the mean and (b) variance of the sampled s_{ij}.

While the distribution of s_{ij} is approximated as a Gaussian distribution, the decision-making error can be calculated as follows. Define a variable τ as:

$$\tau \triangleq \frac{E[Z]}{\sqrt{\text{Var}[Z]}} \simeq \frac{c_3 d\sqrt{N}}{2^k \sigma_{\text{BB}} \sqrt{m}}, \tag{9}$$

where $Z = s_{ij,\text{dep}} - s_{ij,\text{indep}}$. The decision error ε is given by $1 - \Phi(\tau)$, Φ is the cumulative standard Gaussian distribution. If we consider a fixed error rate of $\frac{1}{m}$, N can be obtained as:

$$N \geqslant c_4 2^{2k} \frac{\sigma_{\text{BB}}^2}{d^2} m \log m. \tag{10}$$

Therefore, for a fixed error rate of the interaction model, the population size or the number of individuals that must be evaluated by the fitness function in the proposed approach is $O(m\log(m))$ or $O(n\log(n))$. This result is verified empirically in Section 6.

The differences between this population size bound and one that reported in [43] are: (1) this derivation is based on the nonlinearity instead of the mutual information and (2) as DSM is not constructed using a selected population, the selection pressure is not included in this formulation. Both of these bounds scale as $O(m\log(m))$.

After evaluating all the individuals in the population, pair-wise dependencies (s_{ij} values) should be computed (see Fig. 3). This should be done by a single pass through all the individuals in the population and updating four f' values for each pair of variables. This means computing the f' values involves $O(n^2)$ iterations for each individual. So this computation takes $O(n^2 \times N)$ iterations. While the number of individuals N is bounded by $O(n\log(n))$, the computational cost of calculating the pair-wise dependencies is $O(n^3\log(n))$. During DSM construction, a two-means algorithm is employed to find the threshold that is used to convert the DSM to the binary domain. Maximum number of iterations of the main loop of two-means algorithm is experimentally observed to be less than n. Each iteration of the two-means algorithm involves $O(n^2)$ operations, so finding the best threshold takes $O(n^3)$ operations. Hence, in overall, DSM construction is done by $O(n^3\log(n))$ operations, which is the computational cost of calculating the pair-wise dependencies.

In order to analyze time complexity of the proposed DSM clustering algorithm, the four phases are analyzed one by one. Generating the initial clusters in the first phase involves calculating density of each group of variables by O(n^2) and removing the weak variables by O(n^3). In the second phase, building secondary clusters based on initial clusters is comprised of three steps. These three steps are nested in a loop that repeats at most n times. Step one has the complexity of O(n). Step two has the complexity of O(n^2) in the worst case and the time complexity of step three is O(n^2). Therefore, the second phase has the time complexity of O(n^3). The third phase which revises the secondary clusters, needs O($m^2 \times k^2$) iterations where m is the number of clusters and k is the average size of the clusters. Number of clusters m is equal to n/k, so this phase is done in O(n^2). This will be the case for the fourth phase. Therefore, the overall computational complexity of the proposed DSM clustering algorithm is O(n^3), which is the computational cost of constructing the secondary clusters. It should be noted that no fitness evaluation is required in the process of the proposed DSM clustering algorithm.

Therefore, constructing and clustering the DSM, in overall, is done by O(n^3 log(n)) operations. Empirical results support these theoretical analyses.

6. Experimental results

In this section we present some empirical results to experimentally evaluate the proposed approach. We have employed two types of problems to make experimental results: the $m - k$ trap [8] as a well-known benchmark function and some test problems with overlapping sub-problems. The $m - k$ trap is an additively separable deceptive function consisting of m concatenated trap functions of k variables. Two types of sub-problems scaling can be considered in the problem decomposition: (1) uniform scaling and (2) exponentially scaling [26]. In the uniform scaling, the fitness contributions of all sub-problems are the same but in the exponentially scaling, each sub-problem contributes to the fitness value by a different exponential share. In our experiments we consider uniform scaling. Equation (11) shows the used trap function, where u is the number of bits with value 1 in each sub-problem:

$$\mathrm{trap}_k(u) = \begin{cases} k & \text{if } u = k, \\ k - 1 - u & \text{otherwise.} \end{cases} \qquad (11)$$

The results are presented in three subsections. First, we investigate quality of the constructed DSM and quality of the interaction model that is built by the clustering algorithm. Some accuracy ratios will be defined for this purpose. Second, efficiency of the proposed model building approach is evaluated by reporting two main criteria: (1) the number of fitness evaluations required for building the interaction model and (2) total run time of the algorithm. We then report the results of finding the optimum solution using the interaction model and compare it with BOA [26]. This evaluation is done using the number of fitness evaluations and the overall computational cost as run time of the algorithm.

6.1. *DSM and accuracy of interaction model*

Experimental results in this paper show that if adequate number of individuals evaluated to construct DSM, the resulting DSM captures correct pair-wise dependencies. Otherwise, wrongly captured or non-captured dependencies in the DSM cause low quality linkage groups.

Figure 15 shows binary DSMs constructed with different number of evaluated individuals for Trap7, an $m - k$ trap function with $k = 7$ and $m = 7$ (size of problem, $n = 49$). DSM in Fig. 15(a) has not captured correct pair-wise dependencies well but as the number of fitness evaluations increases and more individuals are evaluated, the pair-wise dependencies are

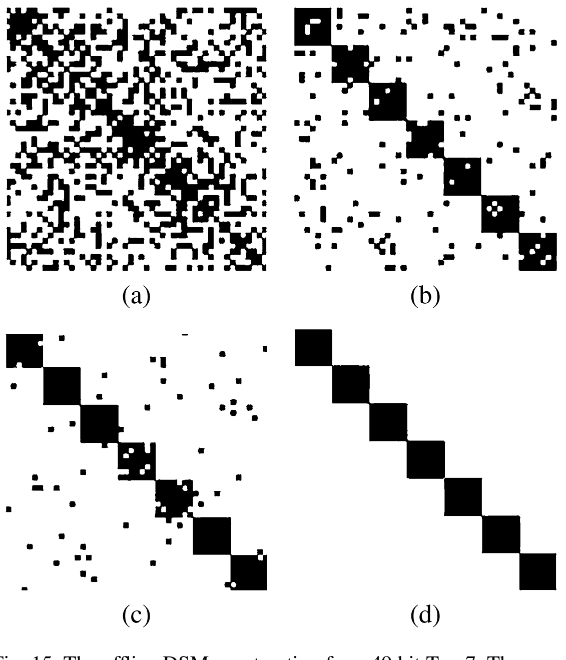

Fig. 15. The offline DSM construction for a 49 bit Trap7. The numbers in parentheses indicate the number of individuals evaluated to construct the DSM. (a) 4690; (b) 32,480; (c) 40,000; (d) 124,000.

captured more accurately. On the other hand, DSM in Fig. 15(d) has captured all the pair-wise dependencies flawlessly; the same observations have also been reported in [40]. Exact clusters on DSM diagonal shown in Fig. 15(d) represent the perfect problem decomposition. Another important observation is that while 40,000 evaluations reveal the approximate decomposition of the problem, more than three times evaluations (124,000) are needed to construct the flawless DSM. So another important factor in the quality of interaction model is the effectiveness of DSM clustering algorithm, which is employed to detect the model from DSM. In other words, if the clustering algorithm is effective and accurate enough, it can extract exact clusters (high quality model) from a less accurate DSM. To investigate this problem, we present some definitions to evaluate the performance of the proposed clustering algorithm precisely.

There are two types of error that can happen when an interaction model represents the genetic interactions [39]. One of them is ignoring a linkage in the model that exists in reality, called *detection failure*. The other one is detecting a linkage that does not exist in reality, called *false interaction*. By calculating these errors, the quality of model can be determined. The first type of error causes BB disruptions that makes the convergence difficult, while the second one did not have such a harmful effect [39]. Since the interaction model consists of some linkage groups, we define a correct interaction group as a group of genes that has no detection failure. In contrast, an incorrect interaction group is one that has at least one detection failure. In our approach, DSM clustering algorithm generates the interaction model. Therefore, *model accuracy* that can be considered as the accuracy of the DSM clustering algorithm is defined in Eq. (12) as the proportion of correct linkage groups in the generated model.

$$\text{Model Accuracy} = \frac{\text{number of correct linkage groups in the model}}{\text{total number of linkage groups}}. \tag{12}$$

The model accuracy will take its maximum value of 1 when the interaction model has no detection failure. Moreover, two other metrics are defined to evaluate the accuracy of constructed DSM and examine the effectiveness of the proposed clustering algorithm. The first metric, named *DSM accuracy*, is defined in Eq. (13) as the ratio of the number of correct linkage groups in the DSM to the total number of linkage groups (that should have been represented in the DSM if it has been constructed flawlessly).

$$\text{DSM Accuracy} = \frac{\text{number of correct linkage groups in the DSM}}{\text{total number of linkage groups}}. \tag{13}$$

The DSM accuracy will be 1 if all the linkage groups represented in the DSM without detection failure. The second metric is called *DSM structural accuracy* (similar to Model Structural Accuracy [19] for Bayesian networks) and is defined in Eq. (14) as the ratio of the number of correct pair-wise linkages captured by DSM to the number of all the linkages captured by it.

$$\text{DSM Structural Accuracy} = \frac{\text{number of correct linkages in the DSM}}{\text{total number of linkages in the DSM}}, \tag{14}$$

where a correct pair-wise linkage is one that exists in reality. DSM structural accuracy will be 1 if all captured linkages are correct. If the constructed DSM flawlessly captures all dependencies and independencies (Fig. 15(d)), both DSM accuracy and DSM structural accuracy will be 1 while in Fig. 15(b) DSM accuracy is $0/7 = 0$ and DSM structural accuracy is about 0.7.

Figure 16 illustrates variation of different accuracy ratios while the number of individuals evaluated for building the interaction model is increased. Experiment is done for a Trap7 function of size 49 with the tight linkages, which will be discussed later in this subsection. All accuracy ratios are averaged over 30 in-

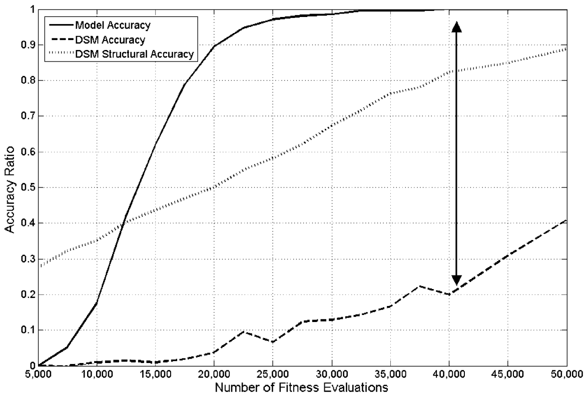

Fig. 16. Variation of accuracy ratios while increasing the number of fitness evaluations (averaged over 30 independent runs).

dependent runs. The solid line in Fig. 16 shows the accuracy of model, obtained by the proposed DSM clustering algorithm when DSM was constructed by increasing the number of fitness evaluations. As it was expected, the model accuracy grew as the number of evaluated individuals increased. Figure 16 also shows the variations of two other metrics while more individuals were evaluated by the fitness function to construct the DSM.

The gap (indicated in Fig. 16 by vertical double arrow line) between the model accuracy and the DSM accuracy at the point where the model accuracy reaches 1, shows the effectiveness of the clustering algorithm in extracting all the linkage groups correctly (a model without detection failures) from a less accurate DSM. The bigger is the gap, the more effective is the clustering algorithm. Although the DSM accuracy shows the ratio of the correct linkage groups captured by DSM, it cannot represent incorrect captured dependencies in the DSM. Since the incorrect captured dependencies may affect the functionality of the clustering algorithm, the DSM structural accuracy is also reported. The DSM structural accuracy increases as the number of incorrect dependencies captured by the DSM decreases. Figure 16 shows that in spite of the existence of incorrect dependencies and non-captured dependencies in DSM (DSM structural accuracy is about 0.8 when the accurate model is extracted), the clustering algorithm can accurately detect all the linkage groups.

In a previously reported comparison between detection metrics that were used to construct DSM [39], simultaneity and mutual information were preferred over nonlinearity. The tested problem was power-law of the OneMax problem and results showed that nonlinearity detects strong interactions between variables while there are no dependencies between variables in The OneMax problem. As it was mentioned before this kind of error (false interaction) is less influential than detection failure. On the other hand, power-law of the OneMax problem can be solved by the proposed approach even with some falsely detected linkages because a GA is employed after building the interaction model.

It should be mentioned that different types of linkages could be found in the problems [41]. In this paper three cases were investigated: *tight*, *loose* and *overlapping*. Interacting genes are arranged next to each other in the tight linkage case. For example, if we let *u(x)* as a counting function that counts the number of 1's in x, the fitness function of an $m-k$ trap function with tight linkages is defined as:

$$\text{trap}_k((u(x_1, x_2, \ldots, x_k)) + \text{trap}_k((u(x_{k+1}, x_{k+2}, \ldots, x_{2k})) + \cdots. \tag{15}$$

In the loose linkage case, interacting genes are away from each other as far as possible and the fitness function is defined as:

$$\text{trap}_k((u(x_1, x_{m+1}, \ldots, x_{(m-1)k+1})) + \text{trap}_k((u(x_2, x_{m+2}, \ldots, x_{(m-1)k+2})) + \cdots. \tag{16}$$

If some interacting genes are the same in the linkage groups, an overlapping case is formed. We have considered two non-cyclic overlapping genes in our experiments and the fitness function for this overlapping case is defined as:

$$\text{trap}_k((u(x_1, x_2, \ldots, x_k)) + \text{trap}_k((u(x_k, x_{k+1}, \ldots, x_{2k-1})) + \cdots. \tag{17}$$

We now demonstrate the validity and generality of our approach by considering other types of linkages. In Fig. 17, the model accuracy is reported on a Trap7 with $m = 7$ with three different linkage cases: tight, loose and overlapping. As it was expected, the model accuracy grew in all cases as the number of fitness evaluations increased and the most accurate model (with no detection failures) is extracted eventually. There were

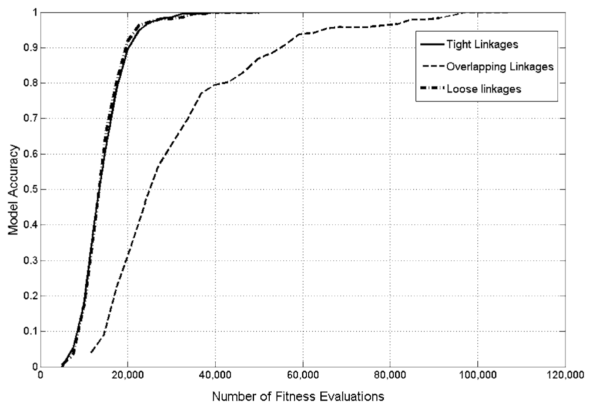

Fig. 17. Model accuracy of Trap7 with $m = 7$ with tight, loose and overlapping linkages (averaged over 30 independent runs).

two other important observations. First, the number of fitness evaluations used to detect all the linkage groups in the tight and loose linkage cases were almost the same. This means that the distance of interacting genes in individuals can not affect the performance of the linkage group detection algorithm. Second, this is not the case for the overlapping linkage. The results show that finding an accurate model in the case of overlapping linkages is more difficult and requires nearly twice the number of fitness evaluations than in the case of tight and loose linkages.

6.2. *Building an accurate model*

In this subsection we evaluate the proposed approach while finding an accurate model in different testing problems. The proposed linkage detection algorithm is evaluated by two performance measures: (1) the number of fitness evaluations needed to detect all the linkage groups and (2) the total running time of the algorithm. The fitness evaluations are needed for the DSM construction phase and there is no need to evaluate the fitness function during DSM clustering. Therefore, the running time of the proposed algorithm is reported to provide the possibility of a fair assessment.

Figures 18 and 19 (log scale) illustrate the number of fitness evaluations needed by DSM construction that leads to maximum model accuracy of more than $(1 - 1/m)$. In our experiments we used three different $m - k$ traps with tight and overlapping linkages: Trap5 ($k = 5$), Trap7 ($k = 7$) and Trap9 ($k = 9$) with different sizes. It should be mentioned that the error rate of the interaction model in all experiments, which are averaged over 30 independent runs, is less than $1/m$. Empirical results show the scalability of algorithm with respect to the required number of fitness evaluations. Theoretical analysis presented in Section 5 shows that the necessary number of evaluated individuals for extracting an accurate model is $\mathrm{O}(n \log(n))$, so does the size of population. Experimental results also match the theory well in all testing problems with tight and even overlapping linkages.

Figures 20 and 21 show total running time in seconds for constructing and clustering DSM. The hardware platform used was an Intel Core2 Duo Processor 2.4 GHz with 2 GB RAM. In Section 5, the time complexity of the proposed linkage detection algorithm was analyzed and it was shown that detecting all linkage groups is done in polynomial time $\mathrm{O}(n^3 \log(n))$. The experiments revealed the scalability of the algo-

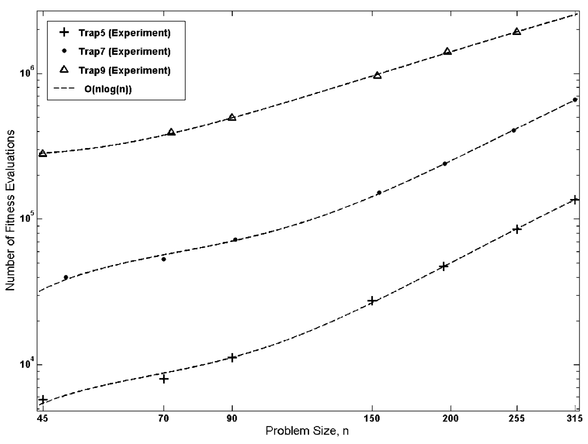

Fig. 18. The number of fitness evaluations needed to build an accurate interaction model in trap functions with tight linkages.

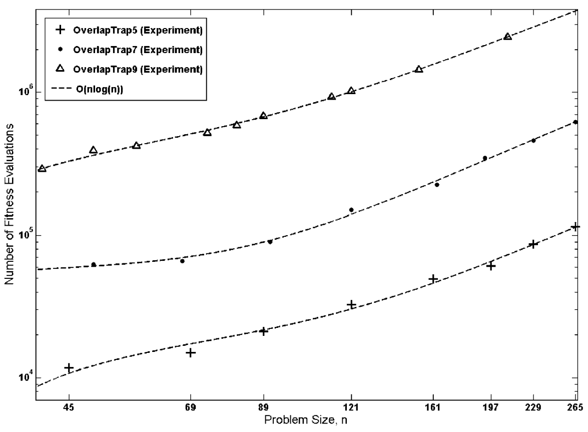

Fig. 19. The number of fitness evaluations needed to build an accurate interaction model in trap functions with overlapping linkages.

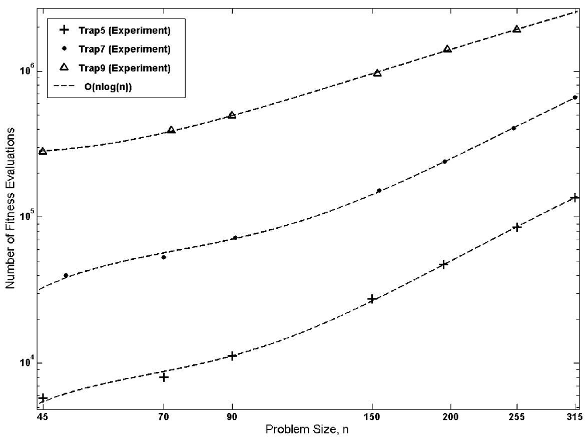

Fig. 20. Run time of the proposed method for different trap functions with tight linkages.

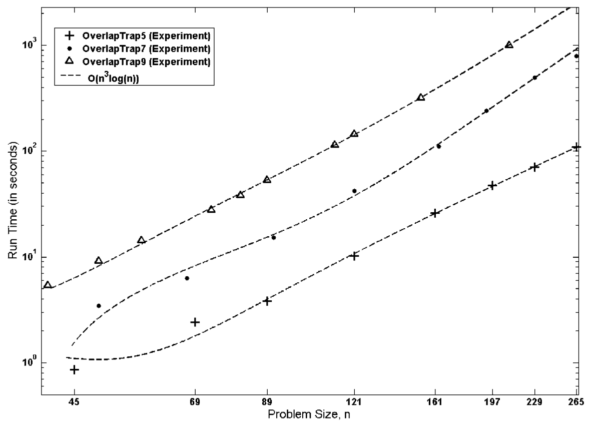

Fig. 21. Run time of the proposed method for different trap functions with overlapping linkages.

rithm in problems with different sizes, different orders of difficulty and different linkage cases. It should be noted that the bottleneck of the proposed linkage group detection algorithm is calculating the pair-wise dependencies and the proposed DSM clustering algorithm does not impose any additional load on the whole process.

It seems that it is useful to compare the proposed approach to the model building approach in DSMGA as the closest research to ours. As it was noted in Section 1, DSMGA utilizes a different clustering algorithm. An MDL-based metric was introduced to evaluate the quality of DSM clustering. Then, a GA was suggested, though not used in practice, to find optimal clustering using the MDL-based metric as fitness function. Instead, a simple hill climber was adopted to find appropriate clustering to save computational time. Another important issue is that the model building approach proposed in this paper is offline while in DSMGA, the interaction model is constructed in each generation. In the offline model building, accuracy of the model is very influential as the success of evolutionary optimization completely depends on model accuracy. This issue was addressed completely in the previous subsection.

As the performance of the DSM clustering algorithm in DSMGA was not reported separately in the published works, a direct comparison is difficult. It should be emphasized that running GA or hill climber and evaluating the MDL-based metric in this method have a considerable computational overhead. While the reported results only include number of fitness evaluations and generation to convergence, this overhead may not be observed. This is the case for some other EDAs like BOA. The model building process that is the main part of these algorithms, involves employing some learning methods and evaluating scoring metrics. Observations in this paper and by other researchers [25] reveal that constructing Bayesian networks in BOA takes more than 95% of the overall run time of the algorithm. But this phenomenon cannot be detected by reporting the generations to convergence, number of fitness evaluations or size of population.

However the only comparable method to our work, called offline utility of DSMGA, has been presented in [40]. The number of individuals that must be evaluated needed to detect the linkage groups of a Trap5 with size 50 with accuracy of 0.998 by the offline utility of DSMGA has been 11,712 (averaged over 100 runs). We carried out the same experiment using the proposed DSM clustering algorithm. The minimum number of fitness evaluations needed to detect all linkage groups by the proposed approach was 7000 (averaged over 100 runs). The resulting DSM accuracy and DSM structural accuracy are reported in Table 1. The difference between DSM accuracy and DSM structural accuracy of the proposed approach and offline utility of DSMGA shows the superiority of the proposed DSM clustering algorithm.

6.3. *Evolutionary optimization using accurate model*

Once the linkage groups information is obtained, this information is used to solve the optimization problem. Any evolutionary algorithm that is capable of incorporating the linkage groups information in the evolutionary process can be used. However, the genetic operators must be changed to be able to use the linkage information. Traditional crossover operators (e.g., uniform, one-point or two-point crossovers) may disrupt BBs while recombining individuals. Now by achieving linkage groups information in this paper we can employ BB-wise crossover [42].

Briefly, the BB-wise crossover is similar to a traditional allele-wise crossover. The difference is that while ordinary crossover mixes genes of some individuals, the BB-wise crossover mixes BBs without disrupting them. BB-wise crossover can be adopted as one point, two point and uniform crossovers. It should be mentioned that the order of BBs has no impact on the functionality of BB-wise crossover because by definition, genes in a BB have no significant interaction with other genes. Because uniform BB-wise crossover is not capable of mixing overlapping BBs well, in the case of non-cyclic overlapping BBs, a population-wise crossover [41] was adopted.

Table 1
The comparison between the proposed method and offline utility of DSMGA [40] on a 50 bits Trap5 function

Method	Number of fitness evaluations	DSM accuracy	DSM structural accuracy	Model accuracy
Proposed method	7000	0.723	0.871	1
Offline DSMGA	11,712	0.906	0.984	0.998

In this subsection we compare empirical results of finding the optimum by detecting linkage groups using the proposed method and feeding them to a GA with BOA. BOA is a well-known optimization algorithm that has been employed to solve a wide range of optimization problems successfully [26]. Bayesian network as a probabilistic model is used in BOA to encode dependencies among variables. After selecting a population of promising individuals, a simple greedy algorithm is used to learn the network based on this population. Usually a scoring metric is employed to evaluate the quality of network structure. Once learning the network is completed, new individuals are generated according to the distribution encoded in the network and incorporated into the population.

In our experiments we used a simple GA without mutation. The tournament selection with tournament size of 4 was adopted as the selection operator in the GA and the BOA. The K2P metric was used for constructing Bayesian network and network parameters were stored in the conditional probability tables while running the BOA. Recent researches [17] revealed that this configuration led to maximum efficiency of BOA. The maximum number of generations in all experiments was set to 150. The minimum population size in all experiments was also determined empirically using the bisection algorithm [26]. The evolutionary process was stopped in both algorithms whenever at least one copy of the optimum was seen in the population. The performance was evaluated using two criteria: (1) the number of fitness evaluations and (2) the total running time of the algorithm. It should be noted that we compared the overall performance of the proposed method with BOA. This means that for example the number of fitness evaluations of the proposed algorithm was the total number of fitness calls for building the model and for finding the optimum (execution of the GA).

Experiments were done on Trap5 and Trap7 with different sizes. Two linkage cases were also tested: tight and overlapping. Figures 22 and 23 show the number of fitness evaluations that were required by BOA and the proposed method with BB-wise crossover GA (the proposed method + B) while solving Trap5 and Trap7 problems with tight linkages respectively. The total run time is reported in Figs 24 and 25.

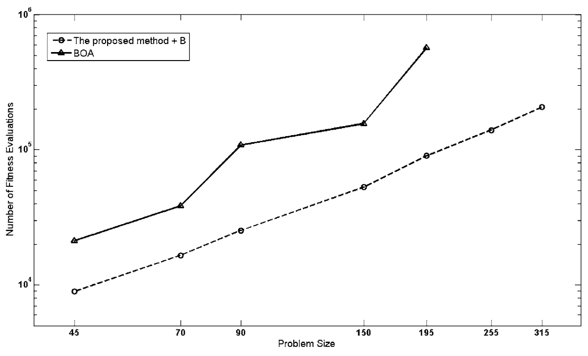

Fig. 22. The comparison between the number of fitness evaluations required by the proposed approach and BOA while solving Trap5 with tight linkages.

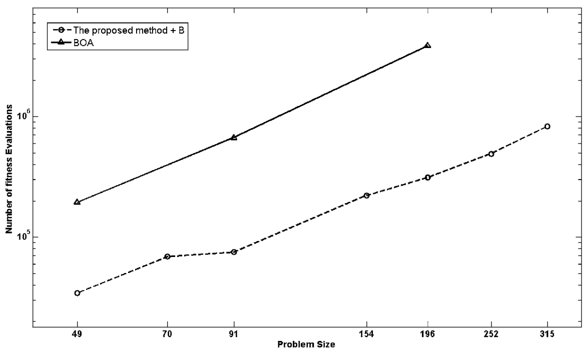

Fig. 23. The comparison between the number of fitness evaluations required by the proposed approach and BOA while solving Trap7 with tight linkages.

The number of fitness evaluations and run time of BOA and the proposed method with population-wise crossover GA (the proposed method + P) on problems with overlapping linkages are illustrated in Figs 26–29. The hardware platform used was an Intel Core2 Duo Processor 2.4 GHz with 2 GB RAM and the reported results were averaged over 30 independent runs.

The reported experimental results showed that the proposed method outperforms BOA with respect to the number of fitness evaluations and the overall run time while solving testing problems. Although it seems that the Bayesian network (in the BOA) is a more general model than the interaction model extracted from DSM (in the proposed method), the experiments revealed that the main portion of computational resources in

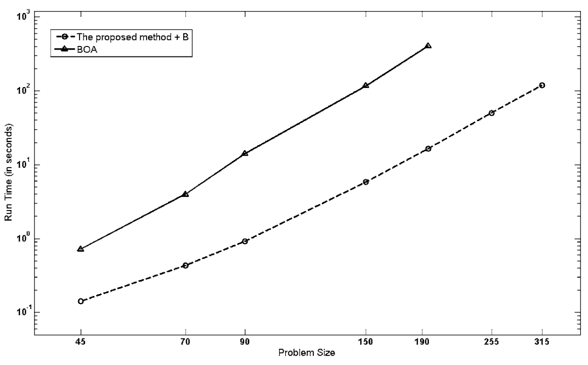

Fig. 24. The comparison between the total run time (time to convergence) of the proposed approach and BOA while solving Trap5 with tight linkages.

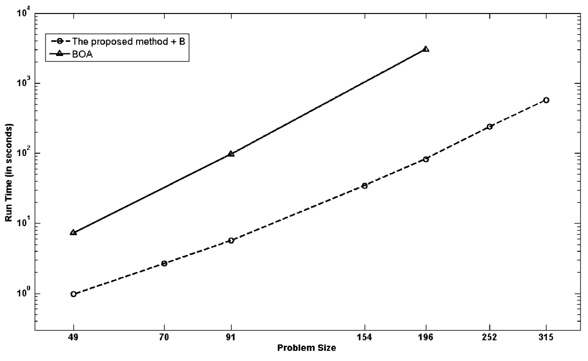

Fig. 25. The comparison between the total run time (time to convergence) of the proposed approach and BOA while solving Trap7 with tight linkages.

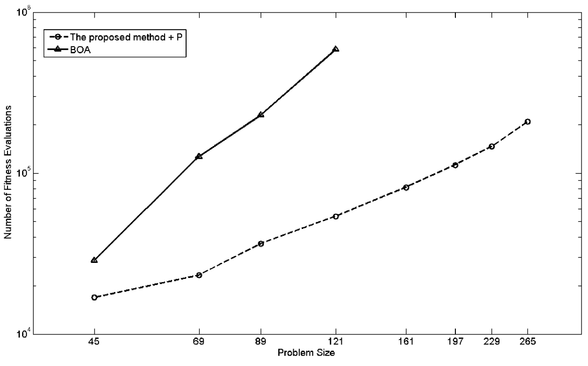

Fig. 26. The comparison between the number of fitness evaluation required by the proposed approach and BOA while solving Trap5 with overlapping linkages.

BOA are consumed to build the Bayesian network. The Bayesian network encodes the dependencies of the variables of the problem and forms the linkage groups. The process of building this model is computationally expensive particularly for the problems with higher or-

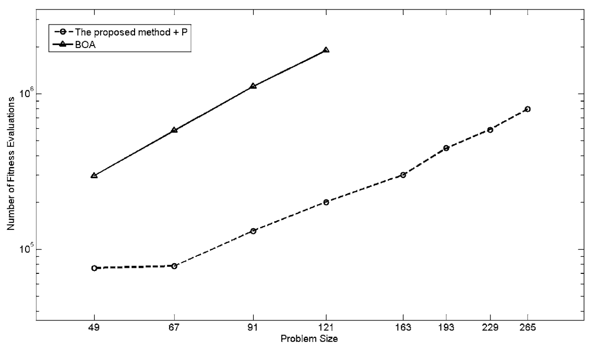

Fig. 27. The comparison between the number of fitness evaluation required by the proposed approach and BOA while solving Trap7 with overlapping linkages.

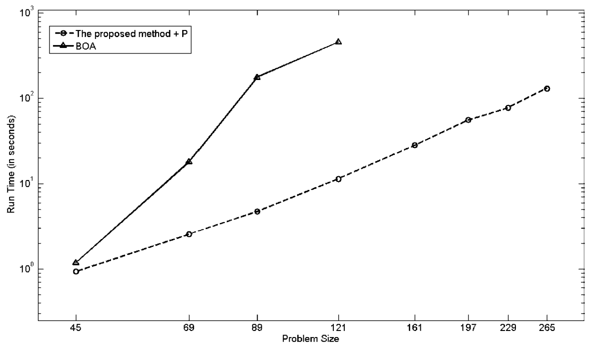

Fig. 28. The comparison between the total run time (time to convergence) of the proposed approach and BOA while solving Trap5 with overlapping linkages.

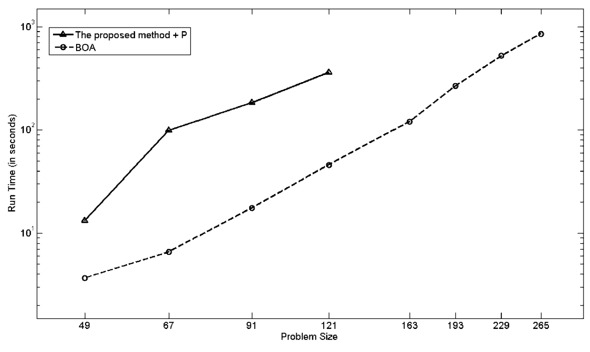

Fig. 29. The comparison between the total run time (time to convergence) of the proposed approach and BOA while solving Trap7 with overlapping linkages.

der of difficulty (large size of BBs). Detecting the linkage groups is also the most computationally expensive part of the proposed method but the theoretical and experimental results showed that the required number of fitness evaluations is bounded by $O(n \log(n))$ and that the overall running time is of $O(n^3 \log(n))$. As it was mentioned previously when the interaction model is obtained accurately it is not hard to find the optimum, even in deceptive problems with tight, loose and overlapping linkages.

7. Conclusions

In this paper, a new linkage group detection algorithm was presented to enhance the efficiency of model building in EDAs. DSM entries representing pair-wise dependencies were calculated by inspecting the fitness changes caused by the perturbation. DSM can capture all pair-wise interactions between variables of the problem if it is constructed using enough number of evaluated individuals. An effective clustering algorithm was then employed to extract the linkage groups from the constructed DSM. The proposed algorithm was shown to be able to detect all linkage groups accurately when the linkages were tight, loose or overlapping. Theoretical analysis showed that the necessary number of evaluated individuals for building an accurate interaction model was bounded by $\mathrm{O}(n \log(n))$, however the experimental results in this paper have supported the theoretical analysis and the total running time of the proposed model building approach scaled as $\mathrm{O}(n^3 \log(n))$. The DSM clustering algorithm did not impose any additional load in terms of fitness evaluations. Experimental results also revealed the scalability of the algorithm in problems with different sizes and different orders of difficulty.

By detecting the linkage information, a simple GA with appropriate crossover was employed to find the optimum of the underlying optimization problem. Comparing the experimental results with another successful evolutionary algorithm, namely BOA, revealed that the proposed method can outperform BOA in testing problems.

References

[1] C. Aporntewan and P. Chongstitvatana, Building-block identification by simultaneity matrix, *Soft Computing – A Fusion of Foundations, Methodologies and Applications* **11**(6) (2007), 541–548.

[2] E. Cantu-Paz, *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic, Boston, MA, 2000.

[3] Y.P. Chen, T.L. Yu, K. Sastry and D.E. Goldberg, A survey of linkage learning techniques in genetic and evolutionary algorithms, Technical Report No. 2007014, Illinois Genetic Algorithm Laboratory, University of Illinois at Urbana-Champaign, 2007.

[4] T.S.P.C. Duque and D.E. Goldberg, A new method for linkage learning in the ECGA, in: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2009)*, Montreal, QC, ACM Press, 2009, pp. 1819–1820.

[5] T.S.P.C. Duque, D.E. Goldberg and K. Sastry, Enhancing the efficiency of the ECGA, in: *Proceedings of Parallel Problem Solving From Nature (PPSN X)*, LNCS, Vol. 5199, Springer, Berlin, 2008, pp. 165–174.

[6] L.R. Emmendorfer and A.T.R. Pozo, Effective linkage learning using low-order statistics and clustering, *IEEE Transactions on Evolutionary Computation* **13**(6) (2009), 1233–1246.

[7] W. Feller, *An Introduction to Probability Theory and Its Applications*, 2nd edn, Wiley, New York, 1971.

[8] D.E. Goldberg, Simple genetic algorithms and the minimal, deceptive problem, in: *Proceedings of Genetic Algorithms and Simulated Annealing*, Pitman, London, 1987, pp. 74–88.

[9] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[10] D. E. Goldberg and S. Voessner, Optimizing global-local search hybrids, in: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 99)*, Morgan Kaufmann, San Francisco, CA, 1999, pp. 220–228.

[11] G. Harik, Linkage learning via probabilistic modeling in the ECGA, Technical Report No. 99010, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 1999.

[12] G.R. Harik, F.G. Lobo and D.E. Goldberg, The compact genetic algorithm, *IEEE Transactions on Evolutionary Computation* **3**(4) (1999), 287–297.

[13] J.A. Hartigen and M.A. Wong, Algorithm AS136: a k-means clustering algorithm, *Applied Statistics* **28**(1) (1979), 100–108.

[14] M. Hauschild, M. Pelikan, K. Sastry and D.E. Goldberg, Using previous models to bias structural learning in the hierarchical BOA, in: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2008)*, Atlanta, GA, ACM Press, 2008, pp. 415–422.

[15] J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[16] H. Kargupta, The gene expression messy genetic algorithm, in: *Proceedings of IEEE International Conference on Evolutionary Computation*, IEEE Press, Nagoya, 1996, pp. 814–819.

[17] H. Karshenas, A. Nikanjam, B.H. Helmi and A. Rahmani, Combinatorial effects of local structures and scoring metrics in Bayesian optimization algorithm, in: *Proceedings of World Summit on Genetic and Evolutionary Computation (GECS 2009)*, Shanghai, China, ACM Press, 2009, pp. 263–270.

[18] P. Larranaga and J.A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Kluwer Academic, Boston, MA, 2002.

[19] C.F. Lima, F.G. Lobo, and M. Pelikan, From mating pool distributions to model overfitting, in: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2008)*, Atlanta, GA, ACM Press, 2008, pp. 431–438.

[20] H. Muhlenbein and G. Paafi, From recombination of genes to the estimation of distributions, I. Binary parameters, in: *Proceedings of Parallel Problem Solving from Nature (PPSN IV)*, LNCS, Vol. 1141, Springer, Berlin, 1996, pp. 178–187.

[21] M. Munetomo and D.E. Goldberg, Linkage identification by non-monotonicity detection for overlapping functions, *Evolutionary Computation* **7**(4) (1999), 377–398.

[22] M. Munetomo and D.E. Goldberg, Identifying linkage groups by nonlinearity/non-monotonicity detection, in: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 99)*, Morgan Kaufmann, San Francisco, CA, 1999, pp. 433–440.

[23] A. Nikanjam, H. Sharifi, B.H. Helmi and A. Rahmani, Enhancing the efficiency of genetic algorithm by identifying linkage groups using DSM clustering, in: *Proceedings of IEEE Congress on Evolutionary Computation (CEC2010)*, Barcelona, IEEE Press, 2010, pp. 1–8.

[24] A. Nikanjam, H. Sharifi, B.H. Helmi and A. Rahmani, A new DSM clustering algorithm for linkage groups identification, in: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2010)*, Portland, OR, ACM Press, 2010, pp. 367–368.

[25] J. Ocenasek and J. Schwarz, The parallel Bayesian optimization algorithm, in: *Proceedings of European Symposium on Computational Intelligence*, Kosice, Slovakia, 2000, pp. 61–67.

[26] M. Pelikan, *Hierarchical Bayesian Optimization Algorithm*, Springer, Berlin, 2005.

[27] M. Pelikan, D.E. Goldberg and E. Cantu-Paz, BOA: the Bayesian optimization algorithm, in: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-99)*, Morgan Kaufmann, San Francisco, CA, 1999, pp. 525–532.

[28] M. Pelikan, K. Sastry and D.E. Goldberg, Sporadic model building for efficiency enhancement of the hierarchical BOA, *Genetic Programming and Evolvable Machines* **9**(1) (2008), 53–84.

[29] M. Pelikan, K. Sastry and D.E. Goldberg, iBOA: the incremental Bayesian optimization algorithm, in: *Proceedings of Genetic and Evolutionary Computation Conference*, Atlanta, GA, ACM Press, 2008, pp. 455–462.

[30] R.P. Salustowicz and J. Schmidhuber, Probabilistic incremental program evolution: Stochastic search through program space, in: *Proceedings of 9th European Conference on Machine Learning (ECML-97)*, LNCS, Vol. 1224, Springer, Berlin, 1997, pp. 213–220.

[31] R. Santana, P. Larranaga and J.A. Lozano, Learning factorizations in estimation of distribution algorithms using affinity propagation, *Evolutionary Computation* **18**(4) (2011), 515–546.

[32] K. Sastry, Evaluation-relaxation schemes for genetic and evolutionary algorithms, Master thesis, General Engineering Department, University of Illinois at Urbana-Champaign, 2001.

[33] K. Sastry and D.E. Goldberg, Let's get ready to rumble: crossover versus mutation head to head, in: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2004),* LNCS, Vol. 3103, Springer, Berlin, 2004, pp. 126–137.

[34] D.M. Sharman and A.A. Yassine, Characterizing complex product architectures, *Systems Engineering* **7**(1) (2004), 35–60.

[35] M. Tsuji and M. Munetomo, Linkage analysis in genetic algorithms, in: *Computational Intelligence Paradigms*, Springer, Berlin, 2008, pp. 251–279.

[36] M. Tsuji, M. Munetomo and K. Akama, Modeling dependencies of loci with string classification according to fitness differences, in: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2004),* LNCS, Vol. 3103, Springer, Berlin, 2004, pp. 246–257.

[37] R.I. Whitfield, J.S. Smith and A.H.B. Duffy, Identifying component modules, in: *Proceedings of Seventh International Conference on Artificial Intelligence in Design (AID'02)*, Springer, Berlin, 2002, pp. 15–17.

[38] A. Yassine, N. Joglekar, D. Braha, S. Eppinger and D. Whitney, Information hiding in product development: the design churn effect, *Research in Engineering Design* **14**(3) (2003), 145–161.

[39] T.L. Yu, A matrix approach for finding extreme: problems with modularity, hierarchy and overlap, PhD thesis, University of Illinois at Urbana-Champaign, 2006.

[40] T.L. Yu and D.E. Goldberg, Dependency structure matrix analysis: offline utility of the dependency structure matrix genetic algorithm, in: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2004)*, LNCS, Vol. 3103, Springer, Berlin, 2004, pp. 355–366.

[41] T.L. Yu, D.E. Goldberg, K. Sastry, C.F. Lima and M. Pelikan, Dependency structure matrix, genetic algorithms, and effective recombination, *Evolutionary Computation* **17**(4) (2009), 595–626.

[42] T.L. Yu, D.E. Goldberg, A. Yassine and Y.-P. Chen, Genetic algorithm design inspired by organizational theory: pilot study of a dependency structure matrix driven genetic algorithm, in: *Proceedings of Artificial Neural Networks in Engineering (ANNIE-2003)*, ASME Press, New York, NY, 2003, pp. 327–332.

[43] T.L. Yu, K. Sastry, D.E. Goldberg and M. Pelikan, Population sizing for entropy-based model building in discrete estimation of distribution algorithms, in: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2007)*, London, ACM Press, 2007, pp. 601–608.