Detecting General Problem Structures with Inductive Linkage Identification

Ying-ping Chen
Yuan-Wei Huang

NCLab Report No. NCL-TR-2009004
March 2009

Natural Computing Laboratory (NCLab)
Department of Computer Science
National Chiao Tung University
329 Engineering Building C
1001 Ta Hsueh Road
HsinChu City 300, TAIWAN
http://nclab.tw/

Detecting General Problem Structures with Inductive Linkage Identification

Ying-ping Chen and Yuan-Wei Huang
Department of Computer Science
National Chiao Tung University
HsinChu City 300, Taiwan
{ypchen, ywhuang}@nclab.tw

March 27, 2009

Abstract

Genetic algorithms and their descendant methods have been deemed robust, effective, and practical for the past decades. In order to enhance the features and capabilities of genetic algorithms, tremendous effort has been invested within the research community of evolutionary computation. One of the major development trends to improve genetic algorithms is trying to extract and exploit the relationship among decision variables, such as estimation of distribution algorithms and perturbation-based methods. In this study, we make an attempt to enable a perturbation-based method, inductive linkage identification (ILI), to detect general problem structures, in which one decision variable can link to an arbitrary number of other variables. Experiments on circular problem structures composed of order-4 and order-5 trap functions are conducted. The results indicate that the proposed technique requires a population size growing logarithmically with the problem size as the original ILI does on non-overlapping building blocks as well as that the population requirement is insensitive to the problem structure consisting of similar sub-structures as long as the overall problem size is identical.

1 Introduction

As practical optimization frameworks, genetic algorithms (GAs) have shown properties of flexibility, robustness, and ease-of-use since they were proposed [1, 2]. These methods usually get good performance when the adopted genetic operators are aware of the relationship among decision variables. Crossover operators in early genetic algorithms are likely to break promising solutions of sub-problems, which are referred to as *building blocks* (BBs) [3]. As a consequence, the overall performance is greatly reduced, or the problem cannot be solved [4]. In order to alleviate this issue, in recent studies, crossover operators or equivalent mechanisms that maintain the structure and diversity of building blocks have been proposed, developed, and examined [5]. These techniques significantly increase the performance of genetic algorithms. To provide the capability of appropriately and effectively handling sub-solutions/building blocks, two key mechanisms, building-block identification and building-block exchange, have to be utilized and integrated in the GA framework. In this study, we focus on the mechanism of building-block identification, generalize the concept regarding the detection of building blocks, and propose the use of a modified version of *inductive linkage identification* [6] to detect general problem structures.

Most of building-block/linkage identifying methods proposed and utilized in previous studies can be broadly classified into the following three categories [7]:

1. Estimation of distribution algorithms;

2. Linkage learning techniques;

3. Perturbation-based methods.

In the first category, estimation of distribution algorithms construct probabilistic models from the selected individuals of the population and describe the relationship among decision variables in a statistical way [8]. Early studies assume no interaction among variables, such as the population-based incremental learning [9] and the compact genetic algorithm [10]. Subsequent researchers use conditional probabilities to capture pairwise and/or multi-variate interactions, e.g., the mutual information maximizing input clustering [11], Baluja's dependency tree approach [12], the bivariate marginal distribution algorithm [13], the factorized distribution algorithm [14], and the Bayesian optimization algorithm [15]. Methods in this category are usually quite efficient from the traditional viewpoint of computational cost in evolutionary computation because they do not need additional fitness evaluations. Nevertheless, less salient building blocks, which contribute little to the total fitness, are less statistically significant and therefore might be ignored and undetected [16].

For the methods of the second category, building-block identification is oftentimes viewed as the (gene/variable) ordering problem. By rearranging variables during the evolutionary process, interdependent variables are put closer according to the adopted coding scheme such that these variables are less likely to be split apart by subsequent operations. In these studies, the messy genetic algorithm [4] and its more efficient descent, the fast messy genetic algorithm [17], exploit building blocks to identify linkages. Since the rearranging mechanism often acts too slow to cooperate with the selection operator, such a condition usually leads to premature convergence. The linkage learning genetic algorithm [18] performs two-point crossover on a specifically designed circular chromosome representation such that tight linkages among related variables can be formed on the chromosome and preserved during the evolutionary process.

Methods in the last category analyze the fitness difference caused by perturbing variables to identify linkages. For example, the gene expression messy genetic algorithm [19] incorporates a special genotype for pairwise relations and a function involving perturbation to find linkage sets. Linkage identification by nonlinear check [7] uses the linear summation of different and non-overlapping building blocks to detect linkages. Borrowing the idea from estimation of distribution algorithms, the dependency detection for distribution derived from the fitness difference [16] clusters variables according to the fitness difference values caused by perturbation. From theoretical points of view, Heckendorn and Wright [20] modeled the linkage/epistatis problem as the Walsh coefficients for Walsh functions and gave a theoretical complexity for perturbation-based methods. Zhou et al. [21, 22] later extended this work from the binary domain to domains of higher cardinalities. Because a perturbed variable only effects the building blocks to which it belongs, information can be obtained on less salient building blocks from fitness difference values. However, since extra fitness evaluations are required when a variable is perturbed, methods in this category cost more function evaluations to identify linkages, although the actual overall computational cost might be less.

From the viewpoint of extracting the problem structure and exploiting the obtained information in order to conduct optimization, estimation of distribution algorithms can be considered as approaches at the "global" end or organizing the obtained information in a "top-down" manner. Estimation of distribution algorithms assume a probabilistic model and adjust the model parameters to fit the promising solutions. On the other hand, linkage learning techniques and perturbation methods are at the "local" end and processing the information in a "bottom-up" manner. These methods implicitly or explicitly extract information out of the selected individuals and recognize the problem structure gradually. In this study, we aim at enhancing the global

problem structure detection capability of perturbation-based methods and at blurring the line between estimation of distribution algorithms and methods in the other two categories.

In particular, we firstly extend the notion of building blocks commonly adopted in perturbation-based methods from overlapping building blocks to general problem structures. Then, a linkage identification technique, called *inductive linkage identification* [6], utilizing the ID3 decision tree [23] is modified and adopted to detect global problem structures. Experiments on the scalability and flexibility are conducted to examine the capability of the modified inductive linkage identification. The results demonstrate that the proposed technique requires a population size growing logarithmically with the problem size. The population requirement is insensitive to the problem structure consisting of similar sub-structures as long as the overall problem size is identical.

For the remainder of this paper, the background of linkage identification is briefly introduced in section 2. Why and how inductive linkage identification works are reviewed with illustrative examples in section 3. Experiments and results are provided in section 4 and discussed in section 5, followed by the summary and conclusions given in section 6.

2 Linkages, Building Blocks, and Problem Structures

De Jong et al. [24] defined the term *dependency*, which is also referred to as *linkage*, as "two variables in a problem are interdependent if the fitness contribution or optimal setting for one variable depends on the setting of the other variable." Moreover, the *order* of a problem is also stated as "the order is the largest number of variables that are interdependent." To obtain the complete information of linkages, the contribution of each possible pair of variables needs to be examined. Although it is usually an expensive work to process all possible pairs of variables, dependencies should be examined as much as possible in a reasonable time such that the employed genetic algorithm can perform well.

The Schema theorem [1] states that short, low-order, and highly fit sub-solutions increase their market shares to be combined. Furthermore, the building block hypothesis [3] implies that combining small partial solutions is essential for genetic algorithms and also consistent with human innovation. According to these observations, a problem model called the *additive decomposable function* (ADF) and written as a sum of low-order sub-functions is proposed in the literature.

Let a binary string of length ℓ, $\mathbf{s} = s_1 s_2 s_3 \dots s_\ell$ present a solution, where $\mathbf{s}$ is a permutation of the decision variables $\mathbf{x} = x_1 x_2 x_3 \dots x_\ell$ determined by the adopted coding scheme. The fitness function for $\mathbf{s}$ is then defined as

$$f(\mathbf{s}) = \sum_{i=1}^{m} f_i(\mathbf{s}_{\mathbf{v}_i}) \ ,$$

where m is the number of sub-functions, $f_i(\cdot)$ is the i-th sub-function, and $\mathbf{s}_{\mathbf{v}_i}$ is the solution string for $f_i(\cdot)$. For example, if $\mathbf{v}_i = (4, 2, 3, 6)$, $\mathbf{s}_{\mathbf{v}_i} = s_4 s_2 s_3 s_6$. If $f_i(\cdot)$ is also a sum of other sub-functions, it can be replaced by these sub-functions. Therefore, without loss of generality, each $f_i(\cdot)$ can be assumed a non-linear function, and the number of variables of $f_i(\cdot)$ is referred to as its order, i.e., complexity. In the ADF model, variables in the same set $\mathbf{v}_i$ are interdependent. These sets referred to as *linkage sets*, and the related term *building block* (BB) is used for candidate solutions to their corresponding sub-functions.

For complex problems, sub-functions are oftentimes overlapping. Similar to interdependent variables, shared variables affect the respective contributions of the overlapping building blocks to the total fitness of the problem and therefore make these building blocks interdependent. Under such a circumstance, considering the interdependent sub-functions as either a single, longer building block or separate, shorter ones become inappropriate. Reviewing previous studies

on pairwise interactions, building blocks, and order-k linkage sets, researchers attempt to capture structures of certain orders. However, if the overall structure can somehow be recognized as that obtained by the model building process in estimation of distribution algorithms, perturbation-based methods should also be able to provide sufficient understandings of the problem for those linkage-aware operations.

Therefore, in this paper, we firstly generalize the concept of overlapping building blocks to the notion of *problem structures* such that interactions among variables can be described as general as possible. The term *sub-problem* is used to describe how the overall problem structure is constructed instead of decomposed. The terms *interaction* and *linkage* are still used for the dependency between any two variables.

3 Inductive Linkage Identification

In this section, a perturbation-based method called *inductive linkage identification* (ILI) is reviewed. Firstly, a brief introduction of ID3 decision trees is given, followed by how ILI adopts the method of ID3 into the fitness perturbation and linkage identification procedure. Then, we describe the modified version of ILI for detecting general problem structures. A simple example is given for illustration.

3.1 ID3 for Recognizing Linkage

The ID3 decision tree construction algorithm is a supervised categorization method working on discrete data sets, in which the datum entries consist of several decision variables and each decision variable is limited to certain predefined values. ID3 aims to build a decision tree according to entropy and information gain. Using a more mathematical description, let a decision variable X have n possible values $\{x_1, x_2, x_3, \ldots, x_n\}$. The entropy of X is defined as:

$$Entropy(X) = -\sum_{i=1}^{n} p(x_i)\ \log(p(x_i))\ ,$$

where $p(x_i)$ is the probabilities for X being x_i. The entropy can be regarded as the indicator for how uniform the contents of a decision variable are. For example, a zero entropy value means that the variable always appears as a certain value, while a high entropy value indicates that there are many distinct values for the variable.

Let Y be also a decision variable with m distinct values $\{y_1, y_2, y_3, ...y_m\}$. If X is somehow affected by Y, it can be split into m subsets for m different values of Y, denoted as X_{y_i}. Based on the entropies, the interference of Y on X is written as the information gain:

$$Gain(X, Y) = Entropy(X) - \sum_{i=1}^{m} \frac{|X_{y_i}|}{|X|}\, Entropy(X_{y_i})\ .$$

Under this definition, higher values are obtained if the respective contents of each X_{y_i} are more identical, meaning that it is more suitable to classify X by Y.

When more decision variables are involved, says that X is affected by Y_1, Y_2, Y_3, ..., etc., the contributions of variables can be decided according to their information gains. By grouping the variables with the highest information gain, the training data set can be split into several subsets. Each subset contains the datum entries with a certain value of that variable. Then, the described procedure is applied on these subsets recursively until all the elements of each subset are identical, respectively. This procedure creates a decision sequence which tells how different values of Y_i result in different x_i values of X. Since this sequence is usually illustrated by using tree structures, the term *decision tree* is used to represent the outcome of this procedure.

	Wind	Outlook	Jogging
1	Strong	Rainy	No
2	Weak	Sunny	No
3	Weak	Cloudy	Yes
4	Weak	Cloudy	Yes
5	Weak	Sunny	No
6	Strong	Sunny	Yes
7	Weak	Cloudy	Yes
8	Strong	Sunny	Yes
9	Strong	Cloudy	No
10	Weak	Rainy	No

(a) Jogging records for 10 days

	Wind	*Outlook*	Jogging
1	Strong	*Rainy*	No
10	Weak	*Rainy*	No
2	Weak	*Sunny*	No
5	Weak	*Sunny*	No
6	Strong	*Sunny*	Yes
8	Strong	*Sunny*	Yes
7	Weak	*Cloudy*	Yes
3	Weak	*Cloudy*	Yes
4	Weak	*Cloudy*	Yes
9	Strong	*Cloudy*	No

(b) Rearranged table by Outlook

Table 1: A example for ID3

A practical example for ID3 might be finding some patterns whether someone will go for jogging under certain weather conditions from records shown in Table 1. In this example, the ID3 proceeds as the following, the three decision variables, Wind, Outlook and Jogging, are denoted respectively as W, O, and J for convenience:

1. Compute $Entropy(J)$:
   $Entropy(J)$
   $= -p(J_{yes}) \log p(J_{yes}) - p(J_{no}) \log p(J_{no})$
   $= -\frac{5}{10} \log \frac{5}{10} - \frac{5}{10} \log \frac{5}{10} = 1.$

2. Compute information gains for W and O. The later is omitted due to the similar calculation:
   $Gain(J, W)$
   $= Entropy(J) - \frac{|J_{W_{Strong}}|}{10} Entropy(J_{W_{Strong}}) - \frac{|J_{W_{Weak}}|}{10} Entropy(J_{W_{Weak}})$
   $= 1 - \frac{4}{10}(-\frac{2}{4} \log \frac{2}{4} - \frac{2}{4} \log \frac{2}{4}) - \frac{6}{10}(-\frac{3}{6} \log \frac{3}{6} - \frac{3}{6} \log \frac{3}{6})$
   $= 1 - 0.4 - 0.6 = 0.$
   $Gain(J, O) = 0.27549$

3. Since $Gain(J, O) > Gain(J, W)$, O is chosen for grouping, and the resultant three subsets are shown in Table 1(b).

4. The same procedure is applied on these three subsets:

   (a) $J_{O_{Rainy}}$: All records give "No," and therefore no need for further split.

   (b) $J_{O_{Sunny}}$ and $J_{O_{Cloudy}}$: Regroup again using decision variable W, respectively. Four subsets are obtained: $J_{O_{Sunny},W_{Strong}}$, $J_{O_{Sunny},W_{Weak}}$, $J_{O_{Cloudy},W_{Strong}}$, and $J_{O_{Cloudy},W_{Weak}}$, respectively give "Yes," "No," "No," and "Yes."

5. Because all subsets give either "Yes" or "No" without exceptions, the classification is finished with the decision tree shown in Figure 1.

This decision tree describes how the records under different weathers can be grouped into classification categories. What the ID3 algorithm does is to construct the relations within those conditions and whether go for jogging. In other words, it identifies dependencies among those decision variables, and what it solves is quite similar to those linkage identification problems.

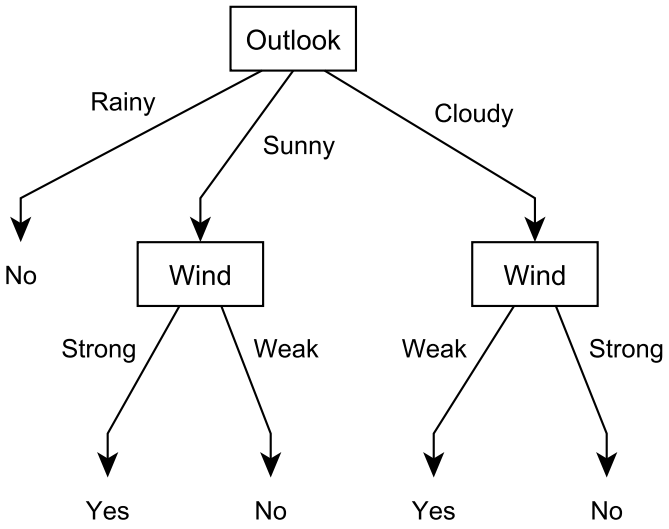


Figure 1: The result decision tree from Table 1

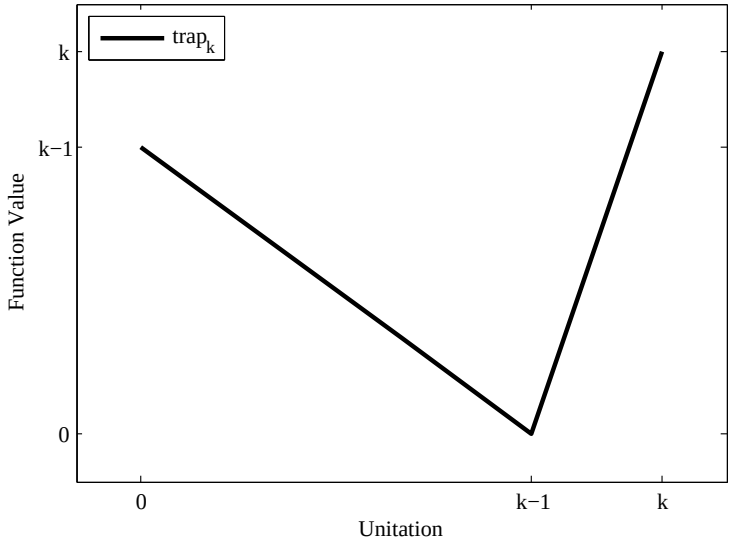


Figure 2: A k-trap function.

For the perturbation procedure, the fitness difference values, denoted as df, are obtained by subtracting the fitness value after perturbation from the original fitness value. This operation implicitly isolates the affected portions of the whole problem structure and reveals them as fitness difference values. The k-trap function [25, 26] is employed in this study as an illustrative example as well as the elementary sub-problem for composing larger problem instances:

$$trap_k(s_1 s_2 s_3 \dots s_k) = \begin{cases} k, & \text{if } u = k; \\ k - u - 1, & \text{otherwise.} \end{cases},$$

where u is the number of 1's in the solution string. Figure 2 shows the characteristic of a k-trap function.

With k-trap functions as elementary sub-problems, more complicated problem instances can be created following the ADF model. For example, an 8-bit function composed of a 3-trap and

$\overline{s_1}s_2s_3\ldots s_8$		f	df
111	01001	5	3
111	10100	5	3
111	01111	3	3
000	11111	7	1
000	00100	5	1
000	00001	5	1
000	10110	3	1
000	11100	3	1
001	01000	4	1
001	00011	3	1
001	00011	3	1
001	10100	3	1
010	01000	4	1
010	00100	4	1
010	01100	3	1

$\overline{s_1}s_2s_3\ldots s_8$		f	df
010	10100	3	1
010	00111	2	1
010	11011	1	1
100	00000	5	-1
100	00100	4	-1
110	11111	5	-1
110	01101	1	-1
110	01111	0	-1
110	11011	0	-1
101	10000	3	-1
101	01101	1	-1
101	11110	0	-1
011	00001	3	-3
011	00110	2	-3
011	01111	0	-3

Table 2: Results obtained by perturbing variable s_1.

a 5-trap function can be defined as

$$f(s_1s_2s_3\ldots s_8) = trap_3(s_1s_2s_3) + trap_5(s_4s_5s_6s_7s_8)\ .$$

By conducting perturbation on a binary variable s_1, the fitness difference df is obtained as

$$\begin{aligned} df &= f(s_1s_2s_3\ldots s_8) - f(\overline{s_1}s_2s_3\ldots s_8) \\ &= (trap_3(s_1s_2s_3) + trap_5(s_4s_5s_6s_7s_8)) \\ &\quad - (trap_3(\overline{s_1}s_2s_3) + trap_5(s_4s_5s_6s_7s_8)) \\ &= trap_3(s_1s_2s_3) - trap_3(\overline{s_1}s_2s_3)\ . \end{aligned} \tag{1}$$

Equation (1) gives a mathematical explanation that df is only affected by the perturbed variable s_1 and those variables belonging to the same sub-problem as s_1. Table 2 is the example of Equation (1) and shows that permutations of s_1, s_2, and s_3 yield the identical df value.

ILI considers the distinct df values as the classification categories and each variable as the decision variable for the ID3 algorithm. By performing ID3 on the perturbed variable as the tree root, a decision tree is accordingly constructed. The internal nodes on the decision tree are then collected as a linkage set $\mathbf{V}_i$. Figure 3 shows the built decision tree corresponding to Table 2, and the internal nodes s_1, s_2, and s_3 form a linkage set.

3.2 Original ILI

The original ILI [6] can handle only those problem structures composed of non-overlapping sub-problems. After perturbing a variable and constructing a decision tree such as that is shown in Figure 3, a linkage set is identified, and the used variables are removed from the variable set. The procedure of perturbation and decision tree construction is repeated on one of the uncategorized variables until all variables are categorized. Taking Table 2 for example, after $\mathbf{V}_1 = \{s_1, s_2, s_3\}$ is identified, ILI then perturbs an uncategorized variable, say, s_4, and constructs a decision tree with $\{s_4, s_5, s_6, s_7, s_8\}$. Then, one of the remaining uncategorized variables if exists. In this example, the final linkage sets are $\mathbf{V}_1 = \{s_1, s_2, s_3\}$ and $\mathbf{V}_2 = \{s_4, s_5, s_6, s_7, s_8\}$.

When there are no overlapping building blocks, experiments [27] demonstrate that the population size required by ILI to correctly identify building blocks grows sub-linearly with the

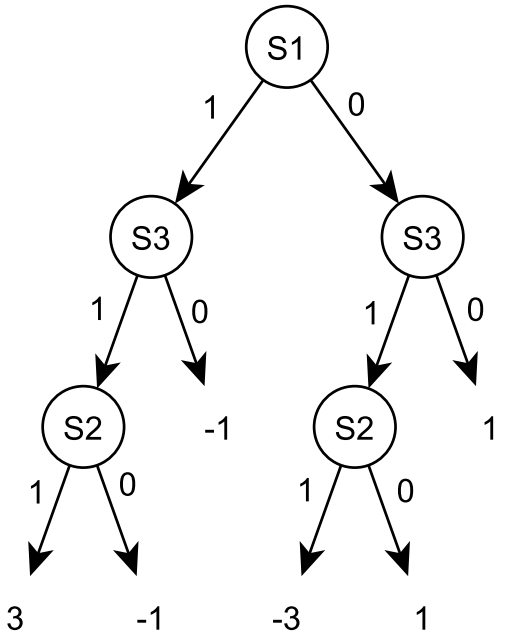


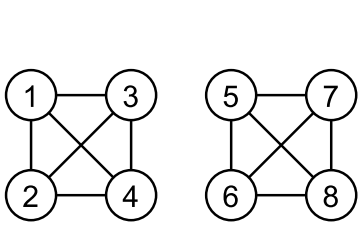
Figure 3: The decision tree constructed for Table 2.

problem size while the complexity of sub-problems is fixed. On the other hand, the population size requirement grows exponentially with the complexity of sub-problem while the problem size is fixed. Such a result indicates that ILI is relatively insensitive to the overall problem size as well as the number of sub-problems and quite sensitive to the complexity of sub-problems.

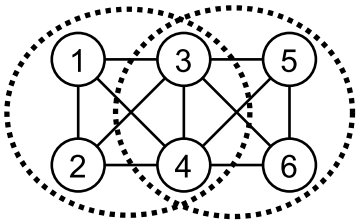
3.3 Modifications on ILI

As mentioned in section 2, overlapping sub-problems may form large, complicated problem structures and may be difficult or inappropriate to be identified as separate building blocks. Taking overlapping sub-problems $trap_4(s_1s_2s_3s_4)$ and $trap_4(s_3s_4s_5s_6)$ as an example, $\{s_1, s_2\}$ indirectly interacts with $\{s_5, s_6\}$ via $\{s_3, s_4\}$ since they belong to both of the sub-problems. These direct and indirect interactions do form a dependency structure of the two sub-problems. Instead of viewing them as either one building block or two, the actual structure should be found and reported to the subsequent linkage-aware operations.

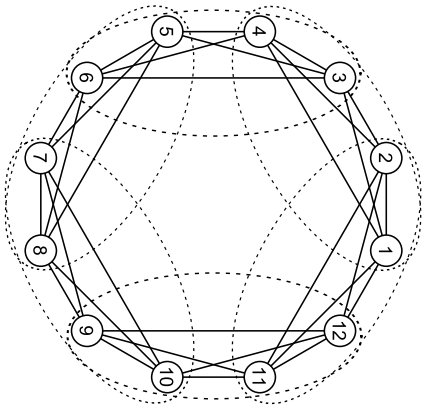
In order to visualize these problem structures, a graph notation is adopted in this paper. Each variable is represented as a graph node, and direct interactions between any two variables are represented as edges between the corresponding graph nodes. For instance, Figure 4(a) shows the graph representation for two non-overlapping $trap_4$ functions. Because variables in the same sub-problems are interdependent, interactions among the four related variables are represented as a complete graph of four nodes. Since these two sub-problems are not overlapping, there exists no edge connecting the two separate sub-graphs. Figure 4(b) is the example for two overlapping sub-functions, $trap_4(s_1s_2s_3s_4)$ and $trap_4(s_3s_4s_5s_6)$, and shows that the shared variables interact with all other variables, while the unshared variables only interacts with the variables of the same sub-problems. Using the graph representation, complex dependency structures can be illustrated. E.g., Figure 4(c) shows a circular structure consisting of six overlapping $trap_4$ sub-problems with two shared variables between adjacent sub-problems. Figure 4(d) is the case



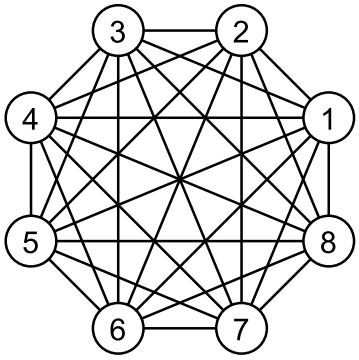
(a) Two non-overlapping $trap_4$ functions.



(b) Two overlapping $trap_4$ functions.



(c) Circular structure.



(d) Complete graph.

Figure 4: Problem structure examples: variables of sub-problems are circled by dashed eclipses.

in which each variable depends on all others to create a very complex overlapping problem structure.

To enable ILI to detect general problem structures composed of arbitrary overlapping sub-problems, key modifications on ILI are proposed in the present work. As described in section 3.2, a variable is removed from the variable set V after being categorized. Such an operation makes the removed variable invisible at later stages of ID3 and thus renders the linkages to other sub-problems undetectable. For example, thinking of Figure 4(b), when the perturbation and ID3 tree construction are performed on variable s_1, the resultant linkage set is $\{s_1, s_2, s_3, s_4\}$, and the rest elements are $\{s_5, s_6\}$ where the relations between $\{s_3, s_4\}$ and $\{s_5, s_6\}$ are lost. One of the proposed modifications is to perturb and perform ID3 on each variable s_i without removing any variable such that all variables can be examined by ID3.

Another modification is to make ILI not directly return linkage sets corresponding to sub-problems, which are also referred to as building blocks. As aforementioned, the concept of building blocks is not very clear when sub-problems are overlapping. In order to determine the overall problem structure, an ℓ-by-ℓ matrix $M_{\ell\times\ell}$ is employed, where ℓ is the number of variables. The element $m_{i,j} = 1$ if there is a connection between variables s_i and s_j; otherwise, $m_{i,j} = 0$. In this study, we adopt undirected linkages. After s_i is perturbed and a linkage set containing s_j is constructed, not only $m_{i,j}$ but $m_{j,i}$ are also marked, although the proposed matrix representation can possibly be used for directed linkages in the future.

Algorithm 1 shows the modified inductive linkage identification procedure. For further illustration, the modified ILI is demonstrated by an example composed of two 4-trap functions with

Algorithm 1 Modified ILI for general problem structures.

```
1: procedure ILI(f, ℓ, n)
2:     Initialize a population P with n strings of length ℓ
3:     Evaluate the fitness of strings in P using f
4:     V ← Shuffle(1, 2, 3, ..., ℓ)
5:     M_{ℓ×ℓ} ← 0_{ℓ×ℓ}
6:     for each v in V do
7:         for each s^i = s^i_1 s^i_2 s^i_3 ... s^i_ℓ in P do
8:             Perturb s^i_v
9:             df^i ← calculate the fitness difference
10:        end for
11:        Build an ID3 tree using (P, df) with v as root
12:        for each internal node v_j in the tree do
13:            m_{v,j} ← 1
14:            m_{j,v} ← 1
15:        end for
16:    end for
17:    Return the structure matrix M
18: end procedure
```

two shared variables defined as

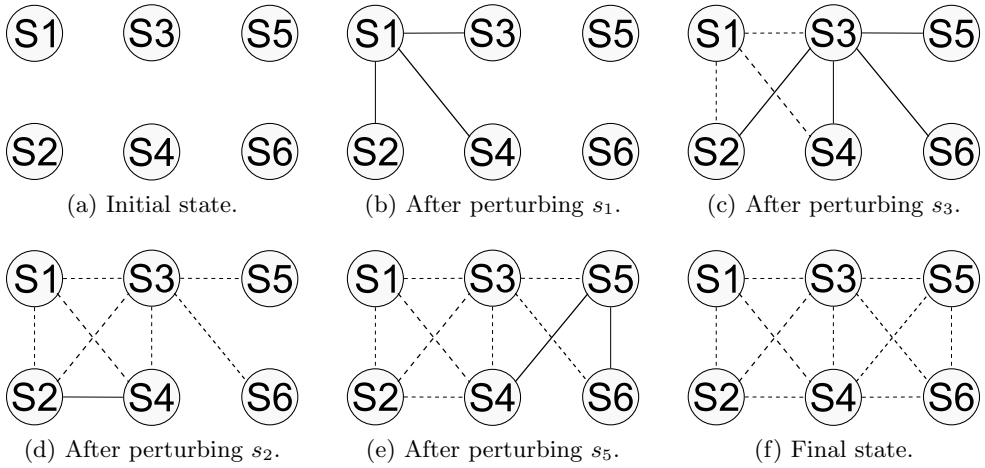
$$f(s_1s_2s_3s_4s_5s_6) = trap_4(s_1s_2s_3s_4) + trap_4(s_3s_4s_5s_6) \tag{2}$$

and shown in Figure 5. Initially, the structure matrix $M_{6\times6}$ is a zero-matrix indicating that there is no known interaction among any variables as shown in Figure 5(a). After initialization, the modified ILI begins to perturb variables in a randomly determined order: s_1, s_3, s_2, s_5, s_4, and s_6. By perturbing and performing ID3 on variable s_1, a linkage set $\{s_1, s_2, s_3, s_4\}$ is recognized and indicates that s_1 interacts with s_2, s_3, and s_4. Figure 5(b) shows the detected partial structure. Notice that although s_2, s_3, and s_4 belong to the same sub-problem as defined in Equation (2), there is no interaction among them detected at the current iteration. Next, when s_3 is perturbed, ID3 identifies that s_3 interacts with all other five variables since it belongs to both sub-problems as shown in Figure 5(c). The procedure is repeated on s_2, s_5, s_4, and s_6 sequentially. After all the variables are proceeded, the final structure is constructed as shown in Figure 5(f).

Because there is no control parameter for the problem order/complexity, the obtained linkage information of the overall problem structure is unconstrained by any assumptions on the complexity of sub-problems. The only key factor in this condition regarding the correctness is whether or not the employed population is large enough for ILI to avoid getting confused by the noise of fitness difference values. Our preliminary experiments involving different problem structures have shown that the proposed ILI modifications are able to construct correct problem structures as long as sufficiently large populations are utilized. For the purpose of gaining more understanding of the population size requirement by the modified ILI, in the next section, we design and conduct more experiments to observe the scalability and flexibility of the proposed modifications.

4 Experiments and Results

Experiments and results on circular structures are examined in this section. Circular structures hold certain good properties for experimental control. The number of linkages increases linearly



(a) Initial state. (b) After perturbing s_1. (c) After perturbing s_3.

(d) After perturbing s_2. (e) After perturbing s_5. (f) Final state.

Figure 5: Problem structures detected during the ILI process. Dashed and solid lines represent known and newly discovered interactions respectively.

with the number of sub-problems and so does the number of nodes. These easily controlled properties enable us to concentrate on the population requirement.

The required population size is determined by a bisection method. For a given problem structure and a range of population sizes $[bound_L, bound_U]$, if the modified ILI can correctly detect the given problem structure for at least 29 times out of 30 independent runs with the population size $P_{size} = (bound_L + bound_U)/2$, we consider that P_{size} is large enough for the modified ILI to detect this problem structure and set P_{size} as the new $bound_U$ for the next iteration. Otherwise, P_{size} is too small to provide appropriate statistics, and thus, the next iteration will be conducted on interval $[P_{size}, bound_U]$. This bisection procedure repeats until the interval is smaller than 2, and the final mean value, P_{size}, is regarded as the required population size. For all the experiments in this study, the bisection process is performed for 50 interdependent trails, and the mean value and the standard deviation are calculated accordingly.

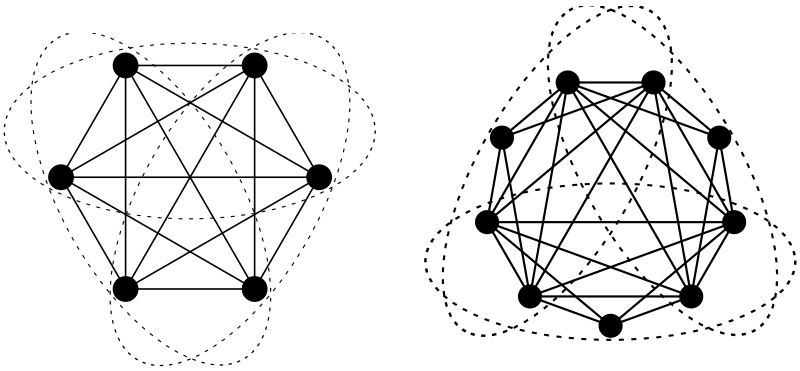
4.1 Scalability on Circular Structures

In this series of experiments, the scalability of the modified ILI is examined by using the $trap_4$ and $trap_5$ functions with circular overlapping problem structures. For the experiments of $trap_4$ functions, each sub-problem shares two variables with one of its neighbor sub-problem and the others two variables with the other neighbor. The circular overlapping structure can be described as

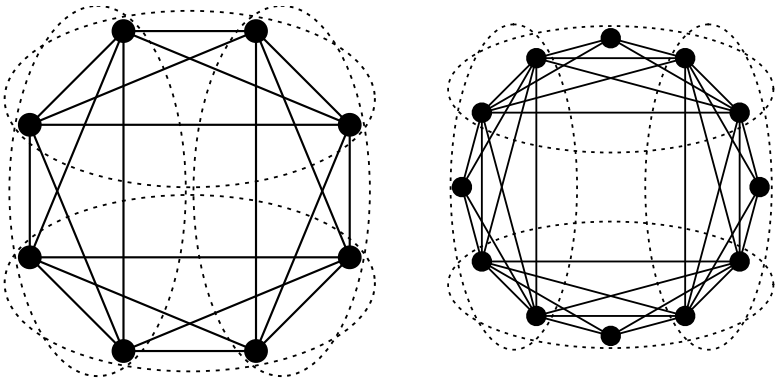
$$C4_n(s_1 s_2 s_3 \dots s_{2n}) = \sum_{i=1}^{n-1} trap_4(s_{2i-1} s_{2i} s_{2i+1} s_{2i+2}) + trap_4(s_{2n-1} s_{2n} s_1 s_2) \ ,$$

where n is the number of sub-problems and greater than 2 to form a circle. For example, $C4_3(s_1 s_2 s_3 \dots s_6) = trap_4(s_1 s_2 s_3 s_4) + trap_4(s_3 s_4 s_5 s_6) + trap_4(s_5 s_6 s_1 s_2)$ is the smallest circular problem structure for $trap_4$ under this definition as shown in Figure 6(a), and inserting one more sub-problem will form a structure shown in Figure 6(c).

The overlapping scheme for $trap_5$ functions is similar, except that each sub-problem has one unshared variable. For example, the minimal circular structure of 3 $trap_5$ functions, shown in



(a) 3 $trap_4$ (b) 3 $trap_5$



(c) 4 $trap_4$ (d) 4 $trap_5$

Figure 6: The minimal circular problem structures.

Figure 6(b), can be put as

$$\begin{aligned} C5_3(s_1s_2s_3\ldots s_9) &= trap_5(s_1s_2s_3s_4s_5) + trap_5(s_4s_5s_6s_7s_8) \\ &\quad + trap_5(s_7s_8s_9s_1s_2)\ , \end{aligned}$$

where s_3, s_6, and s_9 are unshared.

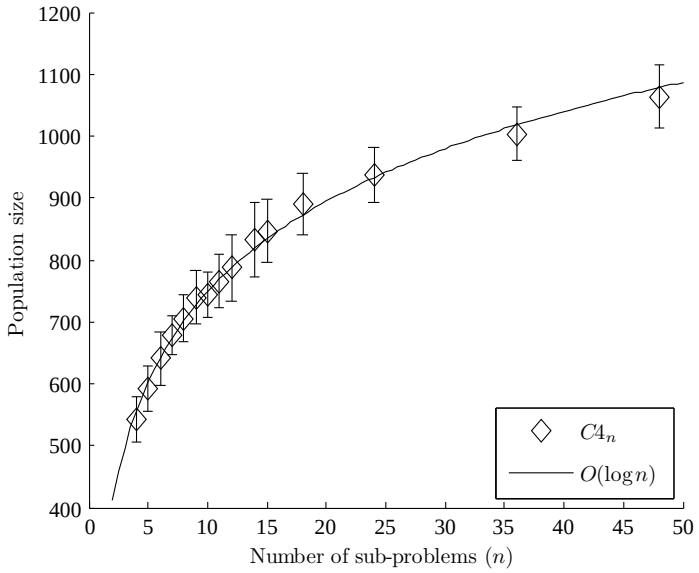
The experimental results of $C4_n$ and $C5_n$ are shown in Figure 7. The results demonstrate that the modified ILI is capable of correctly detecting linkages among variables even when the problem size gets large. The growth rates are both in the logarithmic order.

4.2 Insensitivity on Sub-structures

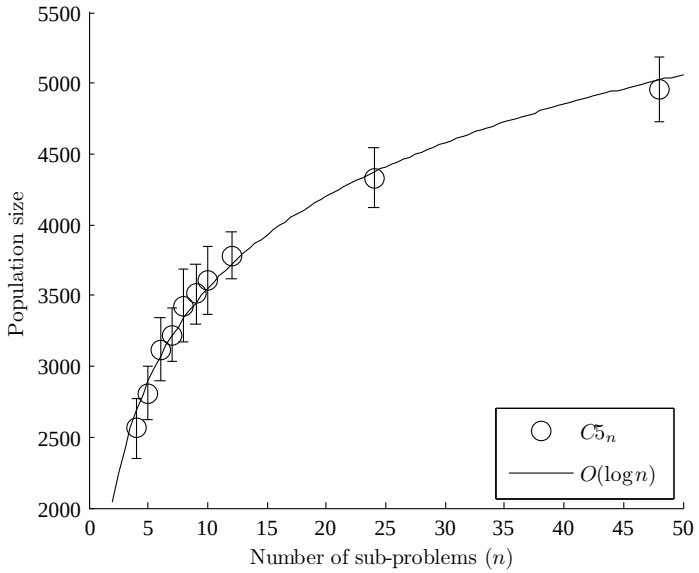
The series of experiments in this section aims to examine the capability of detecting problem structures composed of sub-structures. Separate circular problem structures form a large problem structure, and the population requirement of the modified ILI is compared to that for larger structures of the same problem size. In these experiments, circular problem structures composed of m smaller sub-structures are defined as

$$C4_n^m = \sum_{i=0}^{m-1} C4_n(s_{1+ni}s_{2+ni}s_{3+ni}\ldots s_{2n+ni})\ .$$

in this definition, those sub-problems of a sub-structure are depended in the circular manner, while there exist no dependencies among sub-structures.



(a) $trap_4$ functions as sub-problems.



(b) $trap_5$ functions as sub-problems.

Figure 7: The population requirement for the modified ILI to correctly detect circular problem structures composed of $trap_4$ and $trap_5$ functions.

Figure 8 shows examples of circular structures composed of two and three sub-structures of five $trap_4$ sub-problems. Experimental results on those problem structures, $C4_n$, $C4_n^2$, and $C4_n^3$, are shown in Figure 9.

5 Discussion

In this section, we firstly discuss the experimental results and then the suitability of applying the modified ILI on problems composed of other sub-problems via the ADF model.

5.1 Logarithmic Scalability

From Figure 7, the result for our experiments of the scalability, two observations can be made. The first one is that the results can be well fitted by using logarithmic curves. Such a phenomenon implies that the required population size grows logarithmically with respect to the number of sub-problems and indicates the modified ILI is quite efficient and scalable on the circular problem structures adopted in the experiments. The population size growth rate is also similar to that required by the original ILI on non-overlapping building blocks as reported in the literature [27]. Secondly, since the growth of the required population size can be well fitted with logarithmic curves for both $trap_4$ and $trap_5$ functions, the modified ILI should require similar population sizes for the problems of similar structures.

In our experiment for sub-structures, whose results are shown in Figure 9, the required population sizes of $C4_n^2$ and $C4_n^3$ are very close to that of $C4_n$ when the overall problem sizes are identical. All the experimental results are therefore well fitted by the same logarithmic curve that fits the results of $C4_n$. Such an outcome indicates that the modified ILI is able to correctly identify isolated as well as interdependent parts of a large problem structure without extra cost.

Because the modified ILI is a deterministic method, the number of required function evaluations can be calculated directly according to the required population size. Compare to some theoretical upper bounds, the complexity of the proposed technique empirically obtained is much lower than that proposed in [20] as well as that in [21, 22].

5.2 Other Types of Sub-problems

Since this method is based on the property that the same permutation of a sub-function will cause the same fitness difference under the ADF model, the ID3 algorithm should also work on other functions possessing this property as well as the function adopted in the present work, $trap_k$. In Table 3, we examine this property for two other frequently used functions, $nith_k$ and $valley_k$:

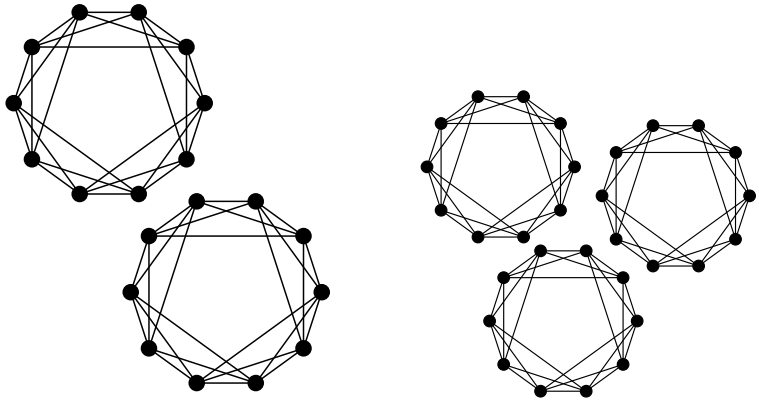
$$nith_k(s_1 s_2 s_3 \dots s_k) = \begin{cases} k, & \text{if } u = k; \\ 0, & \text{otherwise.} \end{cases},$$

$$valley_k(s_1 s_2 s_3 \dots s_k) = |u - \lfloor \frac{k}{2} \rfloor|,$$

where u is the number of 1's in the solution string. As a consequence, the capability of the modified ILI to detect problem structure constructed based on these two frequently used functions are verified.

6 Summary and Conclusions

In this paper, we extended the inductive linkage identification to detect general problem structures composed of overlapping sub-problems and conducted experiments by using circular overlapping structures for gaining more insights and understandings. According to the experimen-



(a) $C4_n^2$ (b) $C4_n^3$

Figure 8: Circular problem structures composed of separate sub-structures, where $n = 5$.

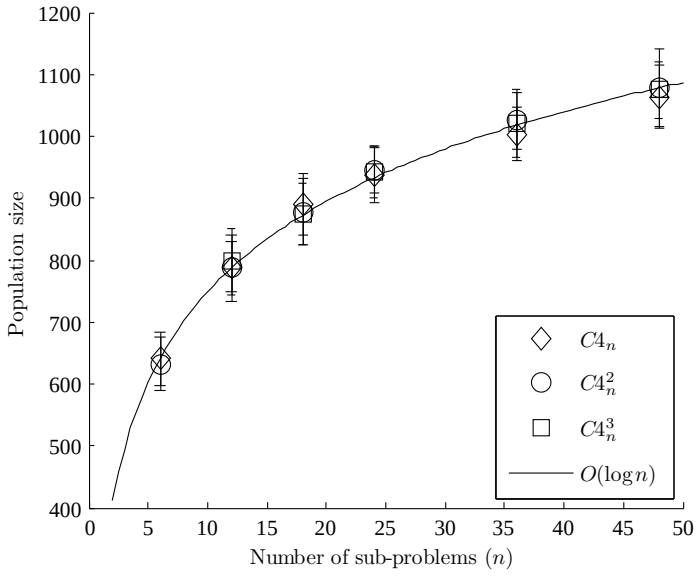


Figure 9: Circular problem structures composed of one, two, and three sub-structures with $trap_4$ as the elementary sub-problems.

$\overline{s_1}s_2s_3\ldots s_7$	f	df
0 0 1 0 0 0 0	0	0
0 0 1 0 0 0 1	0	0
0 0 1 0 1 0 1	0	0
0 0 1 0 1 1 0	0	0
0 1 1 1 1 0 1	0	-4
1 0 0 0 0 0 1	0	0
1 0 0 0 1 1 0	0	0
1 1 0 0 1 0 1	0	0
1 1 0 0 1 0 1	0	0
1 1 0 0 1 1 0	0	0
1 1 0 0 1 1 0	0	0
1 1 0 0 1 1 0	0	0
1 1 1 1 0 0 1	4	4
1 1 1 1 0 1 0	4	4
1 1 1 1 0 1 0	4	4

(a) $nith_4$

$\overline{s_1}s_2s_3\ldots s_7$	f	df
1 0 1 0 0 0 1	1	-1
1 0 1 0 0 1 0	1	-1
1 0 1 0 1 0 0	1	-1
1 0 1 0 1 0 1	0	-1
1 0 1 1 0 0 0	2	1
1 0 1 1 0 0 1	1	1
1 1 0 0 0 0 0	2	-1
1 1 0 0 1 0 1	0	-1
1 1 0 0 1 0 1	0	-1
1 1 0 0 1 1 0	0	-1
1 1 0 0 1 1 0	0	-1
1 1 0 0 1 1 0	0	-1
1 1 1 1 0 0 1	2	1
1 1 1 1 0 1 0	2	1
1 1 1 1 0 1 0	2	1

(b) $valley_4$

Table 3: Fitness differences of different sub-functions by using $f(\mathbf{s}) = func(s_1s_2s_3s_4) + func(s_4s_5s_6s_7)$

tal observations, the proposed technique was found able to correctly detect circular problem structures and require a population size growing logarithmically with the problem size. The population requirement was observed insensitive to the problem structure consisting of similar sub-structures for the identical overall problem size.

One of the major differences between ILI and most of the other existing linkage learning methods is the absence of algorithmic parameters for the complexity of sub-problems. The proposed modifications of ILI keep this feature unchanged. Since ILI performs the task of linkage identification without assumptions on the problem structure, such as the chosen probabilistic model or the maximum degree of interactions, the relationship among variables should be extracted as authentic as possible.

Since the modified ILI is capable of detecting general problem structures, it may be applied in two ways. Firstly, by serving as a preprocessor of genetic algorithms, the proposed technique can express the dependencies among variable as a graph such that delicately-designed genetic, linkage-aware operators, such as CrossNet [5], or certain processing mechanisms may utilize the linkage information to preserve and exchange the building blocks. Secondly, the proposed technique can be employed to inspect and extract the relationship among decision variables for understanding the inner structure of the problem at hand in order to assist potential further applicable operations or be integrated in some advanced frameworks, such as the concept of data-centric grey-box optimization in Optinformatics [28, 29].

Acknowledgments

The work was supported in part by the National Science Council of Taiwan under Grant NSC 96-2221-E-009-196. The authors are grateful to the National Center for High-performance Computing for computer time and facilities.

References

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, April 1992.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[3] ——, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[4] D. E. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithms: Motivation, analysis, and first results," *Complex Systems*, vol. 3, no. 5, pp. 493–530, 1989.

[5] F. Stonedahl, W. Rand, and U. Wilensky, "CrossNet: A framework for crossover with network-based chromosomal representations," in *Proceedings of Genetic and Evolutionary Computation Conference 2008 (GECCO-2008)*, 2008.

[6] C.-Y. Chuang and Y.-p. Chen, "Linkage identification by perturbation and decision tree induction," in *Proceedings of 2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, 2007, pp. 357–363.

[7] M. Munetomo and D. E. Goldberg, "A genetic algorithm using linkage identification by nonlinearity check," in *Proceedings of 1999 IEEE International Conference on Systems, Man, and Cybernetics (SMC '99)*, 1999, pp. 595–600.

[8] H. Mühlenbein and G. Paaß, "From recombination of genes to the estimation of distributions i. binary parameters," in *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature (PPSN IV)*, 1996, pp. 178–187.

[9] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep., 1994.

[10] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 287–297, 1999.

[11] J. de Bonet, C. Isbell, and P. Viola, "MIMIC: Finding optima by estimating probability densities," in *Advances in Neural Information Processing Systems*, M. C. Mozer, M. I. Jordan, and T. Petsche, Eds., vol. 9. The MIT Press, 1997, p. 424.

[12] S. Baluja and S. Davies, "Using optimal dependency-trees for combinational optimization," in *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 30–38.

[13] M. Pelikan and H. Mühlenbein, "The bivariate marginal distribution algorithm," in *Advances in Soft Computing – Engineering Design and Manufacturing*, R. Roy, T. Furuhashi, and P. K. Chawdhry, Eds. London: Springer-Verlag, 1999, pp. 521–535.

[14] H. Müehlenbein and T. Mahnig, "FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions." *Evolutionary Computation*, vol. 7, no. 4, pp. 353–376, 1999.

[15] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "BOA: The Bayesian optimization algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, 1999, pp. 525–532.

[16] M. Tsuji, M. Munetomo, and K. Akama, "Linkage identification by fitness difference clustering," *Evolutionary Computation*, vol. 14, no. 4, pp. 383–409, 2006.

[17] H. Kargupta, "SEARCH, polynomial complexity, and the fast messy genetic algorithm," Ph.D. dissertation, University of Illinois, 1995.

[18] G. R. Harik, "Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms," Ph.D. dissertation, University of Michigan, 1997.

[19] H. Kargupta, "The gene expression messy genetic algorithm," in *Proceedings of the IEEE International Conference on Evolutionary Computation*, 1996, pp. 814–819.

[20] R. B. Heckendorn and A. H. Wright, "Efficient linkage discovery by limited probing," *Evolutionary Computation*, vol. 12, no. 4, pp. 517–545, 2004.

[21] S. Zhou, Z. Sun, and R. B. Heckendorn, "Extended probe method for linkage discovery over high-cardinality alphabets," in *Proceedings of ACM SIGEVO Genetic and Evolutionary Computation Conference 2007 (GECCO-2007)*, 2007, pp. 1484–1491.

[22] S. Zhou, R. B. Heckendorn, and Z. Sun, "Detecting the epistatic structure of generalized embedded landscape," *Genetic Programming and Evolvable Machines*, vol. 9, no. 2, pp. 125–155, 2008.

[23] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, March 1986.

[24] E. D. de Jong, R. A. Watson, and D. Thierens, "On the complexity of hierarchical problem solving," in *Proceedings of Genetic and Evolutionary Computation Conference 2005 (GECCO-2005)*. New York, NY, USA: ACM, 2005, pp. 1201–1208.

[25] D. H. Ackley, *A connectionist machine for genetic hillclimbing*. Norwell, MA, USA: Kluwer Academic Publishers, 1987.

[26] K. Deb and D. E. Goldberg, "Analyzing deception in trap functions," in *Proceedings of the Second Workshop on Foundations of Genetic Algorithms (FOGA 92)*, 1992, pp. 93–108.

[27] C.-Y. Chuang and Y.-p. Chen, "Recognizing problem decomposition with inductive linkage identification: Population requirement vs. subproblem complexity," in *Proceedings of the Joint 4th International Conference on Soft Computing and Intelligent Systems and 9th International Symposium on advanced Intelligent Systems (SCIS & ISIS 2008)*, 2008, pp. 670–675.

[28] M. N. Le and Y. S. Ong, "A frequent pattern mining algorithm for understanding genetic algorithms," in *Proceedings of International Conference on Intelligent Computing 2008 (ICIC 2008)*, 2008.

[29] M. N. Le, Y. S. Ong, and Q. H. Nguyen, "Optinformatics for schema analysis of binary genetic algorithms," in *Proceedings of Genetic and Evolutionary Computation Conference 2008 (GECCO-2008)*, 2008.