Inductive Linkage Identification on Building Blocks of Different Sizes and Types

Ying-ping Chen
Chung-Yao Chuang
Yuan-Wei Huang

NCLab Report No. NCL-TR-2009009
November 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/

Inductive Linkage Identification on Building Blocks of Different Sizes and Types

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

November 29, 2009

Abstract

The goal of linkage identification is to obtain the dependencies among decision variables. Such information or knowledge can be applied to the designs of crossover operators and/or the encoding schemes in genetic and evolutionary methods. Thus, promising sub-solutions to the problem will be less probably disrupted and successful convergences may more likely to be achieved. In our previous studies, a linkage identification technique, called *Inductive Linkage Identification* (ILI), was proposed. This method was established upon the mechanism of perturbation and the idea of decision tree learning. By constructing a decision tree according to decision variables and resulting fitness difference values, the interdependent variables will be determined by the decision tree learning algorithm. In this paper, we aim to acquire more understandings on the characteristics of ILI, especially its behavior under problems composed of different-sized and different-typed building blocks. Experiments showed that ILI can efficiently handle building block of different sizes and is insensitive to building block types. Our experimental observations indicate the flexibility and the applicability of ILI on various elementary building block types that are commonly adopted in experiments.

1 Introduction

Previous studies [1, 2] on genetic algorithms (GAs) have shown that the encoding scheme of solutions is one of the key factors to the success of genetic algorithms. If strongly related variables, which are usually referred to as building blocks (BBs), are arranged loosely for the adopted representation, they are likely disrupted by crossover operations. Such a condition contributes to the drift of population, instead of the convergence toward the optimal solution. Although encoding strongly related variables tightly or making crossover operators aware of such relationships could solve the plight and improve the GA performance [3], both measures require the foreknowledge of the target problem, which is often not the cases when evolutionary algorithms are adopted.

In order to overcome the building block disruption problem, a variety of techniques have been proposed and developed in the past two decades and can be roughly classified into three categories [4]:

1. Evolving representations or operators;
2. Probabilistic modeling for promising solutions;
3. Perturbation methods.

The objective of the techniques in the first class is to make individual promising sub-solutions separated and less likely disrupted by crossover via manipulating the representation of solutions during optimization. Various reordering and mapping operators has been proposed in the literature, such as the messy GA (mGA) [1] and the fast messy GA (fmGA) [5], which is the more efficient descendant of messy GA. The difficulty of these methods is that the reordering operator usually reacts too slow and loses the race against selection and therefore, premature convergence at local optima occurs. Another technique, the linkage learning genetic algorithm (LLGA) [2], uses circular structures for representations with two-point crossover such that the tight linkage might more likely be preserved. While LLGA works well on exponentially scaled problems, it is inefficient when applied to uniformly scaled problems [2].

The methods in the second category are often referred to as the Estimation of Distribution algorithms (EDAs) [6, 7, 8]. These approaches describe the dependencies among variables in a probabilistic manner by constructing a probabilistic model from selected solutions and then sample the built model to generate new solutions. Early EDAs began with assuming no interactions among variables, such as the population-based incremental learning (PBIL) [9] and the compact genetic algorithm (cGA) [10]. Subsequent studies started to model pairwise interactions, e.g., the mutual-information input clustering (MIMIC) [11], Baluja's dependency tree approach [12], and the bivariate marginal distribution algorithm (BMDA) [13]. Multivariate dependencies were then exploited and more general interactions were modeled. Example methods include the extended compact genetic algorithm (ECGA) [14], the Bayesian optimization algorithm (BOA) [15], the factorized distribution algorithm (FDA) [16], and the learning version of FDA (LFDA) [17]. Since model constructing in these methods requires no additional function evaluations, EDAs are usually considered efficient in the traditional viewpoints of evolutionary computation. However, the model constructing mechanism itself is sometimes computationally expensive. The difficulty which EDAs often face is that the lower salience building blocks, which contribute less to the total fitness, are likely ignored rather than captured.

Approaches in the third category observe the fitness differences caused by perturbing variables to detect dependencies. In the literature, the gene expression messy GA (GEMGA) [18] represents the sets of tightly linked variables as weight values assigned to solutions and employs a perturbation method to detect them. GEMGA observes the fitness changes caused by perturbation on every variable for strings in the population and detects interactions among variables according to how likely those variables compose the optimal solution. Assuming that nonlinearity exists within a building block, the linkage identification by nonlinearity check (LINC) [4] perturbs a pair of variables and observes the presence of nonlinearities to identify linkages. If the sum of fitness differences of perspective perturbations on two variables equal to the fitness difference caused by simultaneously perturbing the two variables, the linearity is determined and thus, these two variables are considered independent. Instead of the non-linearity, the descendant of LINC, linkage identification by non-monotonicity detection (LIMD) [19], adopts non-monotonicity to detect interaction among variables. Compared to EDAs, the low salience building blocks are unlikely ignored in these approaches. However, since obtaining fitness differences requires extra function evaluations, perturbation methods are usually regarded demanding more computational efforts to detect linkages. In addition to empirical studies, Heckendorn and Wright generalized these methods through a Walsh analysis [20]. Zhou et al. later extended this study from the binary domain to high-cardinality domains [21, 22].

An interesting algorithm combining the ideas of EDAs and perturbation methods, called *the dependency detection for distribution derived from fitness differences* (D^5), was developed by Tsuji et al. [23]. D^5 detects the dependencies of variables by estimating the distributions of strings clustered according to fitness differences. For each variable, D^5 calculates fitness differences by perturbations on that variable for the entire population and clusters the strings into sub-populations according to the obtained fitness differences. The sub-populations are

examined to find k variables with the lowest entropies, where k is the algorithmic parameter for problem complexity (the number of variables in a linkage set). The determined k variables are considered forming a linkage set. D^5 can detect dependencies for a class of functions that are difficult for EDAs (e.g., functions contain low salience building blocks) and requires less computational cost than other perturbation methods do. However, its major constraint is that it relies on an input parameter k, which may not be available due to the limited information to the problem structure. As a side-effect to the parameter k, D^5 might be fragile in the situation where the problem is composed of subproblems of different sizes.

In our previous work, we proposed *inductive linkage identification* (ILI), a method which is based on the perturbation and integrated with the ID3 [24] algorithm widely used in the field of machine learning. ILI is an unsupervised method without any parameters for the complexity of building blocks, and its scalability and efficiency against the increasing problem sizes have been demonstrated [25, 26, 27]. In this paper, we address more detailed characteristics of ILI in order to gain deeper insights and better understanding of linkage learning. In particular, problems constructed by building blocks of different sizes and sub-functions are studied and experimented upon. Our experimental results indicate that ILI holds the properties of robustness and efficiency when facing various configurations of building blocks.

The reminder of this paper is organized as follows. In section 2, the background of the linkage in GA and the decomposability of problems is briefly introduced. Section 3 gives an introduction of ILI, including a review of the ID3 decision tree learning algorithm, an example illustrating the proposed approach, and an algorithmic description ILI. Section 4 presents the experiments conducted in this study and the results revealing the behavior of ILI. Finally, section 5 summaries and concludes this paper.

2 Linkage and Building Blocks

In this section, we briefly review the definitions and terminologies which will be used through out this paper. As stated in [28], "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," and such relationship between variables is often referred to as *linkage* in the GA literature. In order to obtain the full linkage information of a pair of variables, the fitness contribution or optimal setting of these two variables shall be examined on all possible settings of the other variables.

Although obtaining the full linkage information is computationally expensive, linkage should be estimated by using a reasonable amount of efforts if the target problem is decomposable. According to the Schema theorem [29], short, low-order and highly fit substrings increase their share to be combined. Also stated in the building block hypothesis, GAs implicitly decompose a problem into sub-problems by processing building blocks. It is considered that combining small parts is important for GAs and consistent with human innovation [30]. These lead to a problem model called the *additively decomposable function* (ADF), which can be written as a sum of low-order sub-functions.

Let a string $\mathbf{s}$ of length ℓ be described as a series of variables, $\mathbf{s} = s_1 s_2 \cdots s_\ell$. We assume that $\mathbf{s} = s_1 s_2 \cdots s_\ell$ is a permutation of the decision variables $\mathbf{x} = x_1 x_2 \cdots x_\ell$ to represent the encoding scheme adopted by GA users. The fitness of string $\mathbf{s}$ is then defined as

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

where m is the number of sub-functions, f_i is the i-th sub-function, and $\mathbf{s}_{v_i}$ is the substring to f_i. Each v_i is a vector specifying the substring $\mathbf{s}_{v_i}$. For example, if $v_i = (1, 2, 4, 8)$, $\mathbf{s}_{v_i} = s_1 s_2 s_4 s_8$.

If f_i is also a sum of other sub-functions, it can be replaced by those sub-functions. Thus, each f_i can be considered as a nonlinear function.

By eliminating the ordering property of v_i, we can obtain a set V_i containing the elements v_i. The variables belonging to the same set of V_i is regarded as interdependent because f_i is nonlinear. Thus, we refer to the set V_i as a linkage set. A related term, *building blocks* (BBs), is referred to as the candidate solutions to sub-function f_i. In this paper, only a subclass of the ADFs is considered. We concentrate on non-overlapping sub-functions. That is, $V_i \cap V_j = \emptyset$ if $i \neq j$. In addition, the strings are assumed to be composed of binary variables.

3 Inductive Linkage Learning

In this section, the ideas behind *inductive linkage identification* (ILI) are firstly presented. Then, the ID3 learning algorithm, which is proposed and widely utilized in the field of machine learning, is briefly introduced. An example is given to illustratively explain the mechanism of ILI, followed by the pseudo code.

In ILI, linkage learning is regarded as the issue of decision tree learning. As an illustration, the fitness difference can be derived in the following equation within the ADF model:

$$\begin{aligned} f(s_1 s_2 \cdots s_8) &= f_1(s_1 s_2 s_3 s_4 s_5) + f_2(s_6 s_7 s_8) \\ df_1(\mathbf{s}) &= f(s_1 s_2 \cdots s_8) - f(\overline{s_1} s_2 \cdots s_8) \\ &= f_1(s_1 s_2 s_3 s_4 s_5) + f_2(s_6 s_7 s_8) \\ &\quad - f_1(\overline{s_1} s_2 s_3 s_4 s_5) - f_2(s_6 s_7 s_8) \\ &= f_1(s_1 s_2 s_3 s_4 s_5) - f_1(\overline{s_1} s_2 s_3 s_4 s_5) \ . \end{aligned} \tag{1}$$

Equation (1) indicates that the fitness difference *df* should be affected by only the bits belonging to the same sub-functions as the perturbed bits, which are $s_1 s_2 \cdots s_5$. Since certain fitness difference values are respectively caused by particular bits arranged in some permutation of the sub-function where the perturbed variable belongs, we can consider the task as finding which values of variables will result in the corresponding fitness differences.

We found that this kind of tasks is similar to decision making in machine learning: Giving a condition composed of attributes, an agent (algorithm) should learn to make a decision with the given training samples. When the decision making method is adopted for conduct linkage learning, decision variables are regarded as attributes and the fitness difference values stands for class labels. With this simple and direct mapping, linkage learning in genetic algorithms can potentially be handled with certain well-developed methods in machine learning.

3.1 Decision Tree Learning: ID3

In ILI, the ID3 decision tree learning algorithm [24] is adopted. We consider ID3 as a classification mechanism, and decision learning can be viewed as and achieved by a sequence of classifications. In a classification problem, a training instance is composed of a list of attributes describing the instance and a target value that the decision tree is supposed to predict after training. In our case, as described in section 3.2, the list of attributes is the solution string, and the target value is the fitness difference caused by perturbation.

In its most basic form, ID3 constructs the decision tree in a top-down manner without backtracking. To construct a decision tree, each attribute is evaluated using a statistical property, called the *information gain*, to measure how well it alone classifies the training instances. The best attribute is accordingly selected and used as the root node of the tree. A descendant node of the root is created for each possible value of this attribute, and the training instances are split into appropriate descendant branches. The entire process is repeated by using the training

instances associated with each descendant node to select the best attribute to test at that point of the tree.

The statistical property, information gain, of each attribute is simply the expected reduction in the impurity of instances after classifying the instances with that attribute. The impurity of an arbitrary collection of instances is often called *entropy* in the information theory. Given a collection D, containing instances of c different target values, the entropy of D relative to this c-wise classification is defined as

$$Entropy(D) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i \ ,$$

where p_i is the proportion of D belonging to class i. For simplicity, in all the calculations involving entropy, we define $0 \log_2 0$ to be 0.

In terms of entropy, the information gain can be defined as follows. The information gain, $Gain(D, A)$, of an attribute A relative to a collection of instances D, is

$$Gain(D, A) \equiv Entropy(D) - \sum_{v \in Val(A)} \frac{|D_v|}{|D|} Entropy(D_v) \ ,$$

where $Val(A)$ is the set of all possible values for attribute A, and D_v is the subset of D for which attribute A has value v.

3.2 Exemplary Illustration

This section illustrates the idea that linkage learning is considered as decision learning with an example. We consider a trap function of size k defined as the following:

$$\begin{aligned} f_{trap_k}(s_1 s_2 \cdots s_k) &= trap_k(u = \sum_{i=1}^{k} s_i) \\ &= \begin{cases} k, & \text{if u = k;} \\ k - 1 - u, & \text{otherwise.} \end{cases} \ , \end{aligned}$$

where u is the number of ones in the string $s_1 s_2 \cdots s_k$. Suppose that we are dealing with an eight-bit problem

$$f(s_1 s_2 \cdots s_8) = f_{trap_5}(s_1 s_2 s_3 s_4 s_5) + f_{trap_3}(s_6 s_7 s_8) \ ,$$

where $s_1 s_2 \cdots s_8$ is a solution string. Our goal is to identify the two linkage sets $V_1 = \{1, 2, 3, 4, 5\}$ and $V_2 = \{6, 7, 8\}$.

In the beginning, a population of strings is randomly generated as listed in Table 1. The first column lists the solution strings, and the second column lists the fitness values of the corresponding strings. After initializing the population, we perturb the first variable s_1 ($0 \rightarrow 1$ or $1 \rightarrow 0$) for all strings in the population in order to detect the variables with interdependency on s_1. The third column of Table 1 records the fitness differences, df_1, caused by perturbations at variable s_1.

Then, we construct an ID3 decision tree by using the perturbed population of strings as the training instances and the perturbed variable s_1 as the tree root. Variables in $s_1 s_2 \cdots s_8$ are regarded as attributes of the instances, and the fitness differences df_1 are the target values/class labels. Corresponding to Table 1, an ID3 decision tree showed in Figure 1 is constructed. By gathering all the decision variables on the non-leaf nodes, we can identify a group of s_1, s_2, s_3, s_4, and s_5. As a consequence, linkage set V_1 is correctly identified.

For the remainder of this example, since s_1, s_2, s_3, s_4 and s_5 are already identified as linkage set V_1, we proceed at s_6. The fitness differences after perturbing variable s_6 are showed in

$s_1 s_2 \cdots s_8$	f	df_1
$\overline{0}$0001 011	3	1
$\overline{0}$0001 011	3	1
$\overline{0}$0011 111	5	1
$\overline{0}$0011 111	5	1
$\overline{0}$0100 001	3	1
$\overline{0}$0100 100	3	1
$\overline{0}$0101 111	5	1
$\overline{0}$0110 111	5	1
$\overline{0}$1000 111	6	1
$\overline{0}$1010 011	2	1
$\overline{0}$1101 010	1	1
$\overline{0}$1101 100	1	1
$\overline{0}$1101 101	1	1
$\overline{0}$1110 110	1	1
$\overline{0}$1110 110	1	1
$\overline{0}$1111 001	0	-5
$\overline{0}$1111 110	0	-5
$\overline{0}$1111 110	0	-5
$\overline{0}$1111 111	3	-5
$\overline{1}$0000 010	3	-1

$s_1 s_2 \cdots s_8$	f	df_1
$\overline{1}$0001 010	2	-1
$\overline{1}$0010 101	2	-1
$\overline{1}$0010 110	2	-1
$\overline{1}$0011 000	1	-1
$\overline{1}$0101 010	1	-1
$\overline{1}$0101 100	1	-1
$\overline{1}$0101 110	1	-1
$\overline{1}$0110 101	1	-1
$\overline{1}$0111 100	0	-1
$\overline{1}$1001 011	1	-1
$\overline{1}$1001 111	4	-1
$\overline{1}$1010 011	1	-1
$\overline{1}$1011 001	0	-1
$\overline{1}$1011 010	0	-1
$\overline{1}$1100 000	1	-1
$\overline{1}$1100 010	1	-1
$\overline{1}$1100 011	1	-1
$\overline{1}$1110 101	0	-1
$\overline{1}$1111 010	5	5
$\overline{1}$1111 011	5	5

Table 1: Population perturbed at s_1

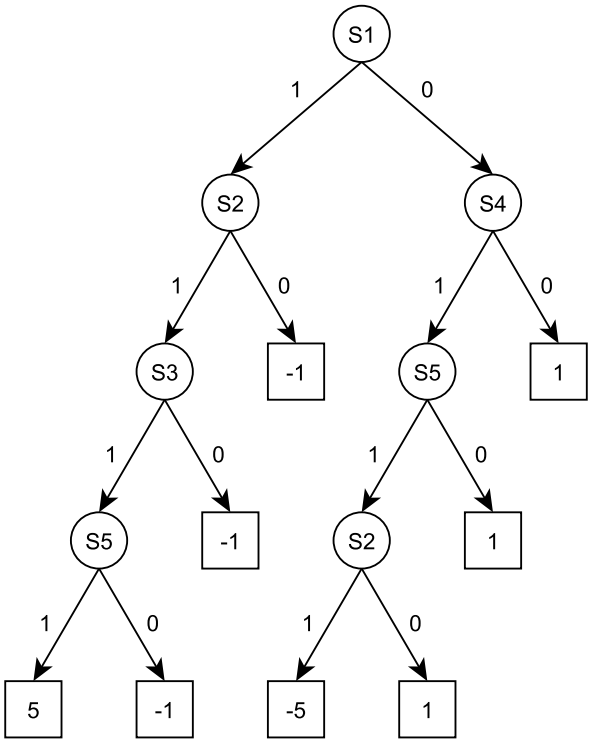


Figure 1: ID3 decision tree constructed according to Table 1

Algorithm 1 Inductive Linkage Identification

```
procedure IdentifyLinkage(f, ℓ)
    Initialize a population P with n string of length ℓ.
    Evaluate the fitness of strings in P using f.
    V ← {1, ..., ℓ}
    m ← 0
    while V ≠ ∅ do
        m ← m + 1
        Select v in V at random.
        V_m ← {v}
        V ← V − {v}
        for each string s^i = s^i_1 s^i_2 ··· s^i_ℓ in P do
            Perturb s^i_v.
            df^i ← fitness difference caused by perturbation.
        end for
        Construct an ID3 decision tree using (P, df).
        for each decision variable s_j in tree do
            V_m ← V_m ∪ {j}
            V ← V − {j}
        end for
    end while
    return the linkage sets V_1, V_2, ···, V_m
end procedure
```

Table 2. Conducting the same procedure, an ID3 decision tree presented in Figure 2 is obtained. By inspecting the decision tree, we obtain variables s_6, s_7, and s_8, which form linkage set V_2. Because all the decision variables are classified into their respective linkage sets, the linkage detecting task is accomplished. ILI finally reports two linkage sets, $V_1 = \{s_1, s_2, s_3, s_4, s_5\}$ and $V_2 = \{s_6, s_7, s_8\}$.

As illustrated in the example, the mechanism of ILI can detect size-varied building blocks without assumptions. Such an ability implies that ILI should be capable of finding all relations among these variables as long as the population size is sufficiently large to provide the statistical significance.

3.3 Inductive Linkage Identification: ILI

In this section, the idea demonstrated in the previous section is formalized as an algorithm, which is called *inductive linkage identification* (ILI) and presented in Algorithm 1. ILI includes mainly the following three steps:

1. Calculate fitness differences by perturbations;
2. Construct an ID3 decision tree;
3. Examine the decision tree to obtain a linkage set.

The three steps repeat until all the variables of the objective function are classified into their corresponding linkage sets.

ILI starts at initializing a population of strings. After initialization, ILI identifies one linkage set at a time using the following procedure: (1) a variable is randomly selected to be perturbed;

$s_1 s_2 \cdots s_8$	f	df_6
11100 $\overline{0}$00	1	0
10011 $\overline{0}$00	1	0
11011 $\overline{0}$01	0	0
01111 $\overline{0}$01	0	0
00100 $\overline{0}$01	3	0
11111 $\overline{0}$10	5	0
10101 $\overline{0}$10	1	0
11100 $\overline{0}$10	1	0
10001 $\overline{0}$10	2	0
11011 $\overline{0}$10	0	0
10000 $\overline{0}$10	3	0
01101 $\overline{0}$10	1	0
00001 $\overline{0}$11	3	-3
00001 $\overline{0}$11	3	-3
11010 $\overline{0}$11	1	-3
11001 $\overline{0}$11	1	-3
11111 $\overline{0}$11	5	-3
11100 $\overline{0}$11	1	-3
01010 $\overline{0}$11	2	-3
10111 $\overline{1}$00	0	0

$s_1 s_2 \cdots s_8$	f	df_6
10101 $\overline{1}$00	1	0
01101 $\overline{1}$00	1	0
00100 $\overline{1}$00	3	0
10010 $\overline{1}$01	2	0
10110 $\overline{1}$01	1	0
11110 $\overline{1}$01	0	0
01101 $\overline{1}$01	1	0
01110 $\overline{1}$10	1	0
01111 $\overline{1}$10	0	0
01110 $\overline{1}$10	1	0
10101 $\overline{1}$10	1	0
01111 $\overline{1}$10	0	0
10010 $\overline{1}$10	2	0
00011 $\overline{1}$11	5	3
00011 $\overline{1}$11	5	3
01000 $\overline{1}$11	6	3
00101 $\overline{1}$11	5	3
11001 $\overline{1}$11	4	3
00110 $\overline{1}$11	5	3
01111 $\overline{1}$11	3	3

Table 2: Population perturbed at s_6

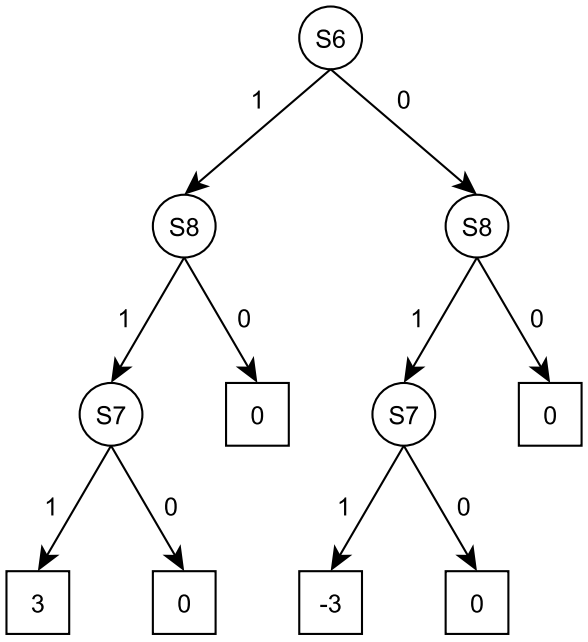


Figure 2: ID3 decision tree constructed according to Table 2.

(2) an ID3 decision tree is constructed according to the fitness differences caused by perturbations; (3) by inspecting the constructed tree, the variables used in the decision tree are collected and considered as a linkage set.

As clearly showed in Algorithm 1, there are no parameters needed for the complexity of sub-functions. That is, ILI does not rely on any assumption on the size of building blocks while other existing perturbation methods usually requires the maximum size of building blocks to be specified. This property of being unsupervised distinguishes ILI with other existing methods. The only factor effecting the correctness of ILI is whether or not the solution strings in the population can provide sufficient information for the ID3 construction.

In our previous studies [25, 26], we already know that the required population size growths linearly with the problem size while the building blocks size is constant. Those results indicate that ILI is more efficient than LINC($\mathcal{O}(\ell^2) = \mathcal{O}(k^2m^2)$) [4] and $D^5(\mathcal{O}(\ell) = \mathcal{O}(km))$ [23]. In order to gain further understanding on the flexibility and applicability of ILI, in the next section, experiments focusing on the building blocks of different sub-functions as well as lengths are conducted and discussed.

4 Experiments and Results

Experiments and results of ILI on binary ADFs are presented in this section. These experiments are designed to gain more understandings of the behavior of ILI on problems of different sub-function compositions, including size-varied, size-mixed building blocks and different sub-functions.

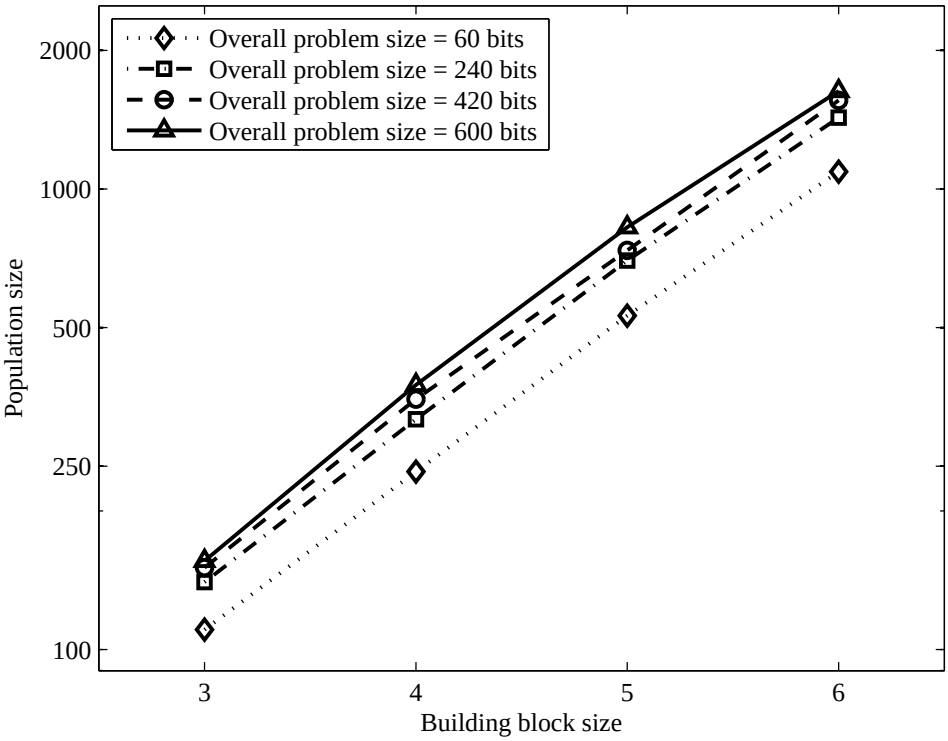
The required population size reflects the behavior of ILI. We determine the population requirement in a bisection manner: For a given problem and a possible population size ranging from P_L to P_U, firstly population size $P = (P_L + P_U)/2$ will be configured for ILI. If ILI can correctly identify all the building blocks within the problem for 30 consecutive and independent runs, P will be regarded sufficiently large for the problem. The next iteration will perform on the range $[P_L, P]$. Otherwise, the range $[P, P_U]$ will be used. This procedure repeats until the range is small than a predefined distance, which is 2 in this study, and the last examined population size is considered as the minimal requirement for the test problem.

4.1 Different Building Block Sizes

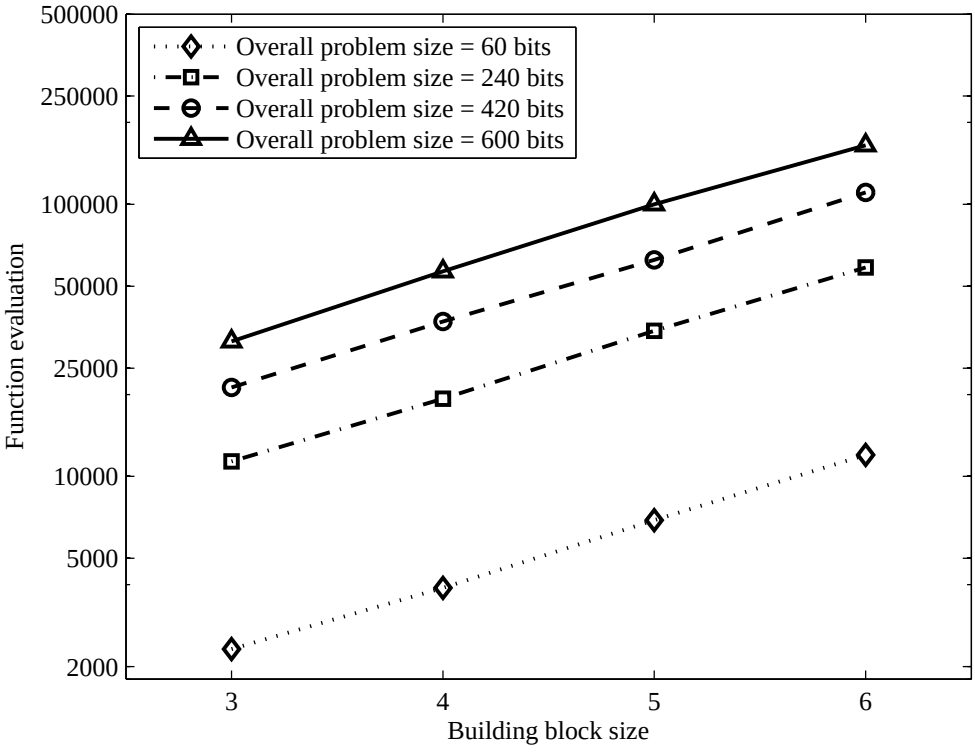
This section describes the experiment on the problems of the same overall size but with different-sized sub-functions. From our experimental results with different configurations of the building block size k and the number of build blocks m, we group those results with the overall problem sizes and arrange them with the building block size k. Thus, the results of the same problem size with different k can be examined.

Figures 3(a) and 3(b) show the experimental results where the overall problem sizes are 60 bits, 240 bits, 420 bits, and 600 bits with a log-scaled y-axis. The strait lines indicate that for identical overall problem sizes, the requirements of both the population size and the function evaluation growth exponentially.

With the exponential regression of the experimental results, an estimation of $y = C \times 2^{a \times k}$ can be obtained, where a is a constant around 0.55 and C varies with different problem sizes. Earlier studies of LINC [4] and Heckdorn's work [20] respectively suggested an empirical and a theoretical upper bounds of function evaluations, which are both in the form of $2^k \ell^j \log(\cdot)$ for problems of ℓ bits, composed of order-k building blocks and each building block sharing j bits with others. Comparing our empirical results with the upper bounds, ILI shows the same computational complexity of the exponential growth with k. However, the regression gives 0.55



(a) Population size



(b) Function evaluation

Figure 3: Requirements on different building block sizes

as the base of exponent and thus indicates the practically better efficiency than the suggested upper bound when the complexity of sub-problem increases.

4.2 Mixed Building Block Sizes

One of the key features of ILI is unsupervised. In this section, we inspect this feature by conducting experiments on the problems consisting of non-overlapping building blocks of several sizes as

$$trap_{k+\ell}(\cdot) = \sum_{i=1}^{m} (trap_k(\cdot) + trap_\ell(\cdot)) \ ,$$

where m is the number of $trap_k$ and $trap_\ell$. By designing the experiments in this way, the empirical results can therefore be easily compared with those from problems consisting of identical sub-problem complexities in the following manner: For each problem size obtained from the experiment of $trap_{k+\ell}(\cdot)$, two results of the same amount of $trap_k$ and $trap_\ell$ from experiments in section 4.1 are summed up to get the same problem size and total number of building blocks, interpolation is utilized when there is no results of such configurations. These calculated numbers are denoted as $trap_k + trap_\ell$ in Figure 4 with the experimental results $trap_{k+\ell}$.

Firstly, these results show that ILI is capable of detecting building blocks of different sizes within one problem without any extra information regarding the complexity of sub-problems. Secondly, comparing with calculated data, it can be seen that although ILI requires more function evaluations for the problems composed of mixed building block sizes, the growth rate is still linear or very close to linear. The observation indicates that identifying size-varied building blocks within a problem poses no particular difficulties for ILI. Such a property of robustness makes ILI more practical when being applied to real world problems where information regarding the sub-problem complexity is usually unavailable and no guideline exists to make appropriate assumptions.

4.3 Building Blocks of Various Elementary Functions

Despite of using $trap_k$ functions as the sub-function to construct building blocks, the capability of ILI to handle building blocks formed by other functions showed in Figure 5 is examined in this section. These elementary functions are used to compose the objective function according to the ADF model, and the complexity of order 4 is adopted in this section.

Figure 6 shows the experimental results. The required population sizes and function evaluations of $trap_4$, $nith_4$, $tmmp_4$, and $valley_4$ are plotted together, and the standard deviation of the results for $trap_4$ is also shown in the figures. Because the population and function evaluation requirements of these problems are similar, the behavior of ILI should also be similar for problems constructed by mixing sub-problems of the same complexity. Moreover, the applicability of ILI on a wide range of problems is also confirmed. ILI is capable of detecting the interactions among decision variables as long as a sufficiently large population is employed to provide significant statistics.

5 Summary and Conclusions

In this paper, we examined ILI on several different configurations of building blocks in order to gain better understandings. We focused on the mixed sizes of building blocks and the elementary functions of different types. These series of experiments verified the efficiency of ILI on the population requirement growth, the robustness of ILI on the mixed sizes of building blocks, and the applicability of ILI on building blocks formed with various elementary functions.

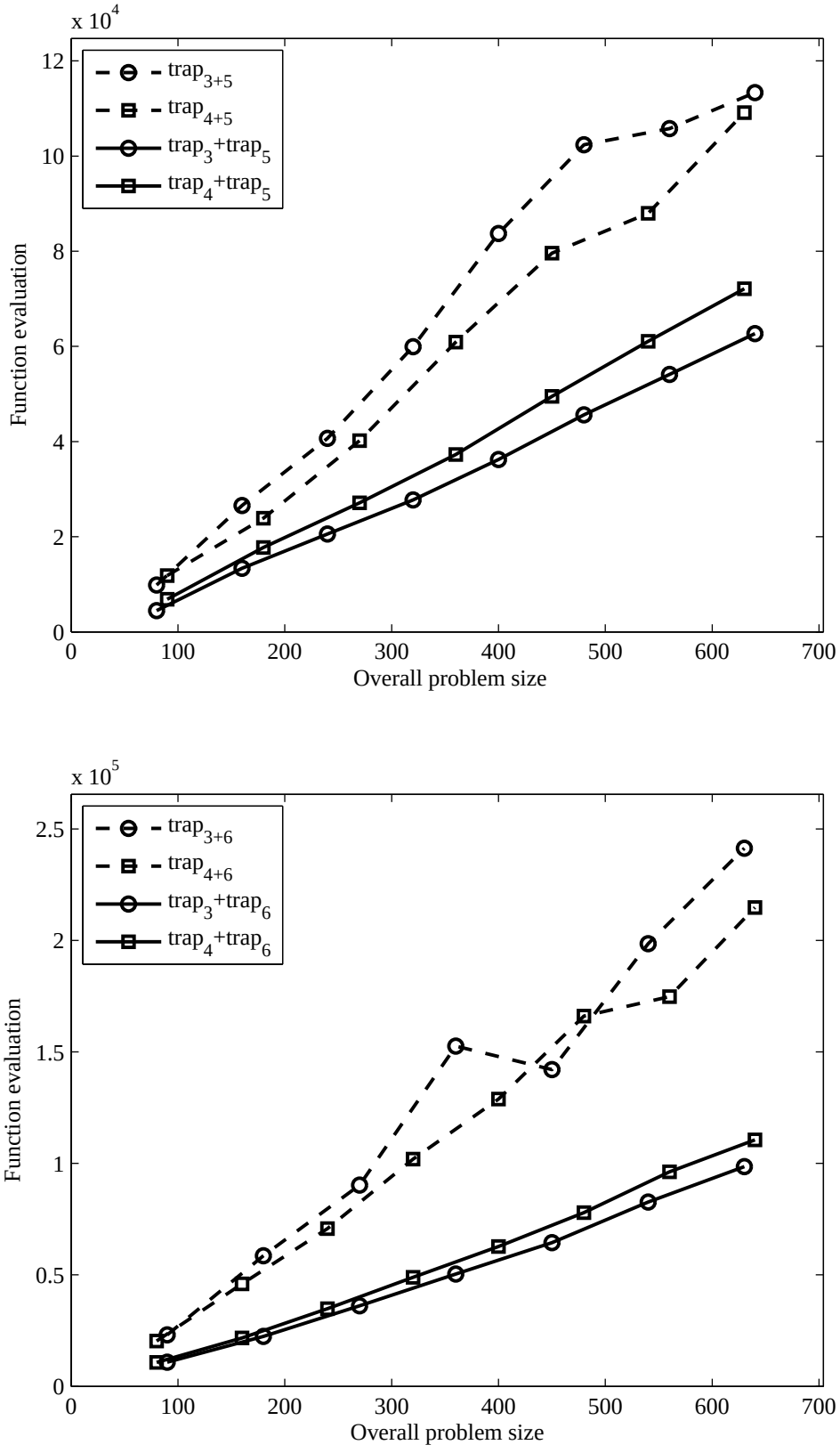
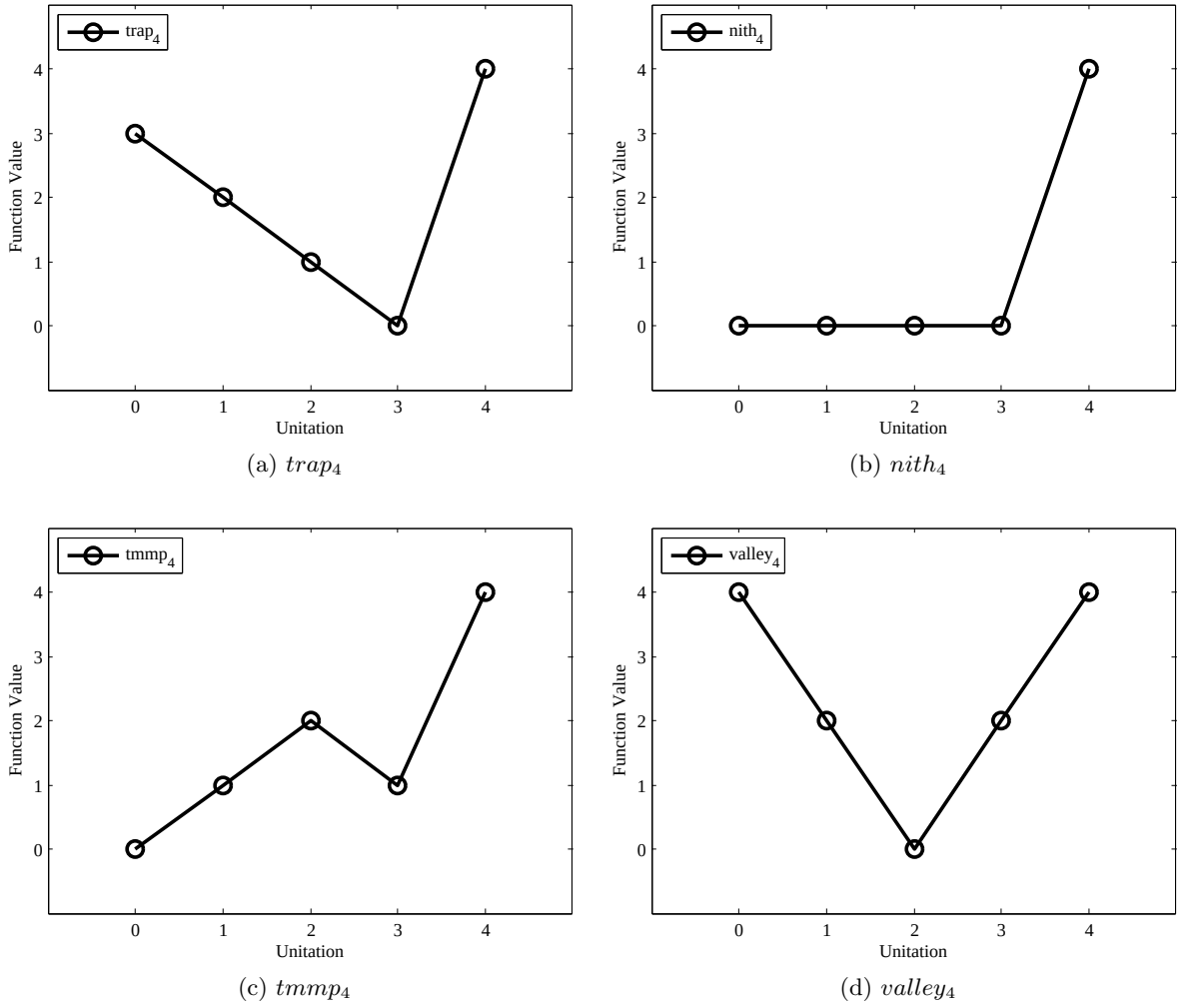


Figure 4: Problems with mixed building block sizes. The solid lines represent the actual experimental results while the dashed lines are the summed up calculations from section 4.1.



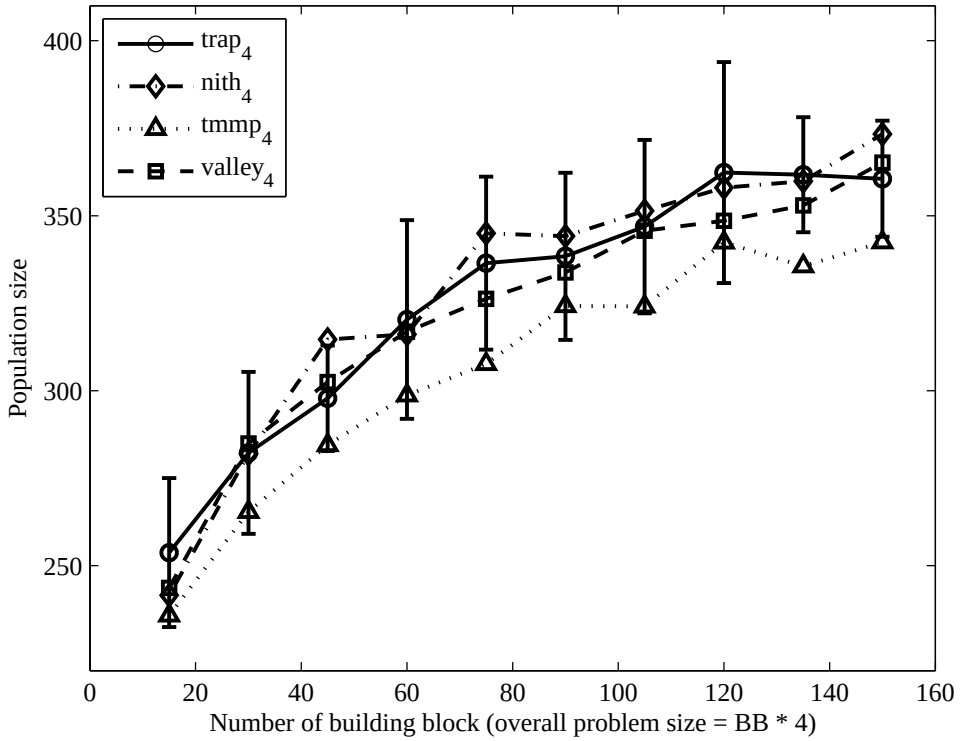
(a) $trap_4$ (b) $nith_4$

(c) $tmmp_4$ (d) $valley_4$

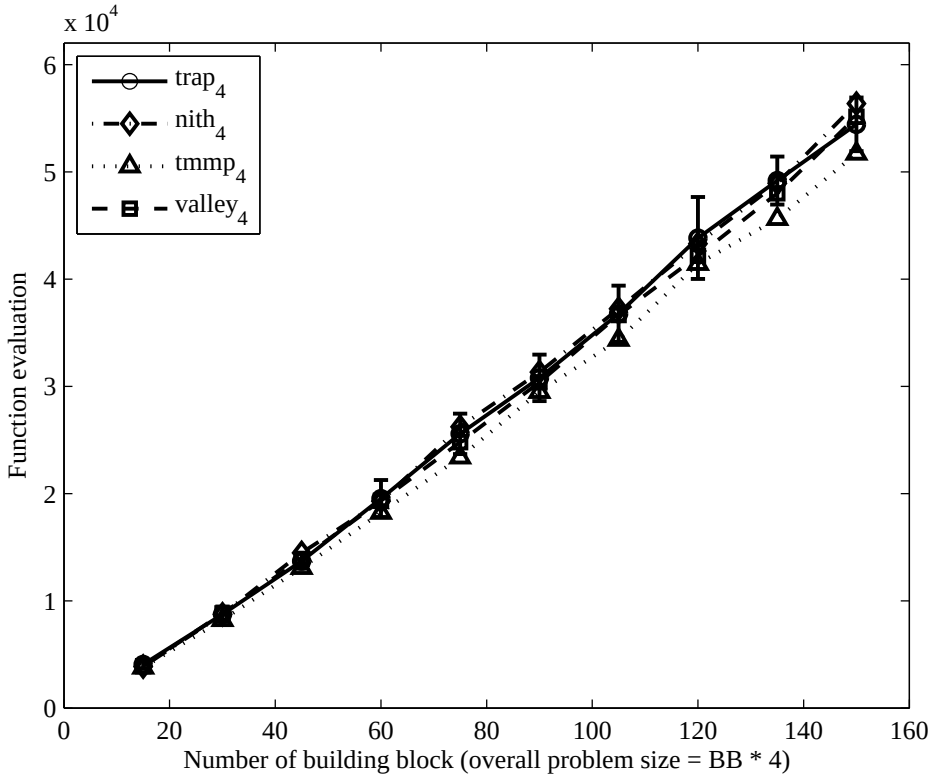
Figure 5: Elementary functions adopted in the series of experiments in section 4.3.

From the experiments of building block sizes, it is demonstrated that the required function evaluations grow exponentially with the size of building blocks when the overall problem size remains constant. Such a result is consistent with the conclusions of previous studies from other researchers in the manner of Big-$\mathcal{O}$ while ILI demands less computational resource in practice. Another observation is that when ILI performs on problems composed of mixed-sized building blocks, the computational complexity of ILI is still in the same order. This phenomenon indicates that detecting these more complicated problem structures poses no particular difficulty for ILI. Finally, the experimental results obtained by using four different elementary functions to construct building blocks are quite similar. Thus, this series of experiments evidentially prove that ILI behaves similarly when handling sub-problem of different types. As a consequence, we can now know that the most important factor that affects ILI's ability to identify linkage is the size of building blocks. ILI is insensitive to other factors commonly studied by the related work, including the overall problem size, the number of building blocks, the scaling of building blocks, and the type of building blocks. Hence, ILI can be considered as a good linkage learning technique and can be adopted as a tool for analyzing structures of target problems or a pre-processing procedure in frameworks of genetic algorithms.

Since its introduction, ILI as a linkage learning technique has been empirically proven efficient, robust, and widely applicable. Research along this line includes integrating ILI into a



(a) Required population sizes



(b) Required function evaluations

Figure 6: Experimental results on different building block types.

GA framework, handling real-world applications with ILI, exploring ILI's capability of analyzing problem structures, and understanding the nature of linkage learning via getting deeper insights of ILI. We will continue to work on advancing the knowledge on linkage learning in order to practically help the algorithmic development of genetic algorithms and theoretically reveal the operation principle of evolutionary computation.

Acknowledgments

The work was supported in part by the National Science Council of Taiwan under Grant NSC 98-2221-E-009-072. The authors are grateful to the National Center for High-performance Computing for computer time and facilities.

References

[1] 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.

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

[3] F. Stonedahl, W. Rand, and U. Wilensky, "Crossnet: a framework for crossover with network-based chromosomal representations," in *Proceedings of ACM SIGEVO Genetic and Evolutionary Computation Conference 2008 (GECCO-2008)*, 2008, pp. 1057–1064.

[4] M. Munetomo and D. Goldberg, "Identifying linkage by nonlinearity check," Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, IlliGAL Report No. 98012, 1998.

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

[6] 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.

[7] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, ser. Genetic algorithms and evolutionary computation. Boston, MA: Kluwer Academic Publishers, October 2001, vol. 2.

[8] M. Pelikan, D. E. Goldberg, and F. G. Lobo, "A survey of optimization by building and using probabilistic models," *Computational Optimization and Applications*, vol. 21, no. 1, pp. 5–20, 2002.

[9] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," 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, November 1999.

[11] J. de Bonet, C. Isbell, and P. Viola, "MIMIC: Finding optima by estimating probability densities," in *Advances in Neural Information Processing Systems*, vol. 9, 1997, pp. 424–430.

[12] S. Baluja and S. Davies, "Using optimal dependency-trees for combinational optimization," in *Proceedings of the Fourteenth International Conference on Machine Learning (ICML '97)*, 1997, pp. 30–38.

[13] M. Pelikan and H. Mühlenbein, "The bivariate marginal distribution algorithm," in *Advances in Soft Computing - Engineering Design and Manufacturing*, 1999, pp. 521–535.

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

[15] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "BOA: The Bayesian optimization algorithm," in *Proceedings of Genetic and Evolutionary Computation Conference 1999 (GECCO-99)*, vol. 1, 1999, pp. 525–532.

[16] H. Mühlenbein 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.

[17] H. Mühlenbein and R. Höns, "The estimation of distributions and the minimum relative entropy principle," *Evolutionary Computation*, vol. 13, no. 1, pp. 1–27, 2005.

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

[19] M. Munetomo and D. E. Goldberg, "Identifying linkage groups by nonlinearity/nonmonotonicity detection," in *Proceedings of Genetic and Evolutionary Computation Conference 1999 (GECCO-99)*, vol. 1, 1999, pp. 433–440.

[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, June 2008.

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

[24] J. R. Quinlan, "Induction of decision trees," in *Readings in knowledge acquisition and learning: automating the construction and improvement of expert systems.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 349–361.

[25] 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.

[26] ——, "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.

[27] Y.-W. Huang and Y.-p. Chen, "On the detection of general problem structures by using inductive linkage identification," in *Proceedings of ACM SIGEVO Genetic and Evolutionary Computation Conference 2009 (GECCO-2009)*, 2009, pp. 1853–1854.

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

[29] J. H. Holland, *Adaptation in natural and artificial systems.* Cambridge, MA, USA: MIT Press, 1992.

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