Inductive linkage identification on building blocks of different sizes and types

Ying-ping Chen a, Chung-Yao Chuang a & Yuan-Wei Huang a

a Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan

Published online: 11 Apr 2011.

To cite this article: Ying-ping Chen , Chung-Yao Chuang & Yuan-Wei Huang (2012) Inductive linkage identification on building blocks of different sizes and types, International Journal of Systems Science, 43:12, 2202-2213, DOI: 10.1080/00207721.2011.566639

To link to this article: http://dx.doi.org/10.1080/00207721.2011.566639

PLEASE SCROLL DOWN FOR ARTICLE
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, Taiwan

(Received 24 March 2010; final version received 17 February 2011)

The goal of linkage identification is to obtain the dependencies among decision variables. Such information or knowledge can be applied to design crossover operators and/or the encoding schemes in genetic and evolutionary methods. Thus, promising sub-solutions to the problem will be disrupted less likely, and successful convergence may be achieved more likely. To obtain linkage information, a linkage identification technique, called Inductive Linkage Identification (ILI), was proposed recently. ILI was established upon the mechanism of perturbation and the idea of decision tree learning. By constructing a decision tree according to decision variables and fitness difference values, the interdependent variables will be determined by the adopted decision tree learning algorithm. In this article, we aim to acquire a better understanding on the characteristics of ILI, especially its behaviour under problems composed of different-sized and different-type building blocks (BBs) which are not overlapped. Experiments showed that ILI can efficiently handle BBs of different sizes and is insensitive to BB types. Our experimental observations indicate the flexibility and the applicability of ILI on various elementary BB types that are commonly adopted in related experiments.

Keywords: inductive linkage identification; ILI; linkage learning; BBs; genetic algorithms; evolutionary computation

1. Introduction

Previous studies (Goldberg, Korb, and Deb 1989; Harik 1997) on genetic algorithms (GAs), which are widely utilised to handle control and engineering problems (Wang 2009; Li and Li 2010; Gladwin, Stewart, and Stewart 2011), have shown that the encoding scheme of solutions is one of the key factors to the success of GAs by demonstrating that simple GAs fail to handle problems of which the solutions are represented with loose encodings while genetic algorithms capable of learning linkage succeed. If strongly related variables, which are usually referred to as building blocks (BBs), are arranged loosely with the adopted representation, they are likely to be disrupted by crossover operations. Such a condition contributes to the divergence of population, instead of the convergence towards optimal solutions. Although encoding strongly related variables tightly or making crossover operators aware of such relationships could mitigate the problem and improve the GA performance (Stonedahl, Rand, and Wilensky 2008), both measures require the foreknowledge of the target problem, which is often not the case in which evolutionary algorithms are adopted.

In order to overcome the BB disruption problem, a variety of techniques have been proposed and developed in the past two decades and can be roughly classified into three categories (Munetomo and Goldberg 1998; Chen, Yu, Sastry, and Goldberg 2007):

1. Evolving representations or operators;
2. Probabilistic modelling 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 to be disrupted by crossover via manipulating the representation of solutions during optimisation. Various reordering and mapping operators have been proposed in the literature, such as self-crossover (Pal, Nandi, and Kundu 1998), which is proven able to generate any arbitrary permutation of the symbols, the messy GA (mGA) (Goldberg et al. 1989), and the fast mGA (fmGA) (Kargupta 1995), which is the more efficient descendant of mGA. The difficulty faced by these methods is that the reordering operator usually reacts too slow and loses the race against selection. Therefore, premature convergence at local optima occurs. Another technique, the linkage learning GA (LLGA) proposed by Harik (1997), uses circular structures as the representation with two-point crossover such that the tight linkage might be more likely preserved. LLGA works well while the shares of BBs are exponentially apportioned in the total fitness,
which are usually referred to as exponentially scaled problems. However, it is inefficient when applied to uniformly scaled problems.

The methods in the second category are often referred to as the estimation of distribution algorithms (EDAs) (Mühlenbein and Paß 1996; Larrañaga and Lozano 2001; Pelikan, Goldberg, and Lobo 2002). 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) (Baluja 1994) and the compact GA (cGA) (Harik, Lobo, and Goldberg 1999). Subsequent studies started to model pairwise interactions, e.g. the mutual-information input clustering (MIMIC) (de Bonet, Isbell, and Viola 1997), Baluja’s dependency tree approach (Baluja and Davies 1997), and the bivariate marginal distribution algorithm (BMDA) (Pelikan and Mühlenbein 1999). Multivariate dependencies were then exploited, and more general interactions were modelled. Example methods include the extended compact GA (ECGA) (Harik 1999), the Bayesian optimisation algorithm (BOA) (Pelikan, Goldberg, and Cantú-Paz 1999), the factorised distribution algorithm (FDA) (Mühlenbein and Mahnig 1999) and the learning version of FDA (LFDA) (Mühlenbein and Höns 2005). Since model constructing in these methods requires no additional fitness evaluations, EDAs are usually considered efficient in the traditional viewpoints of evolutionary computation, especially when fitness evaluations involve time-consuming simulations. However, the model constructing mechanism itself is sometimes computationally expensive with a large population size, which usually occurs in evolutionary methods. The difficulty which EDAs often face is that the BBs contributing less to the total fitness are likely ignored rather than recognised.

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) (Kargupta 1996) models the sets of tightly linked variables as weights assigned to solutions and employs a perturbation method to detect them. GEMGA observes the fitness changes caused by perturbations on every variable for strings in the population and detects interactions among variables according to how likely the variables compose optimal solutions. Assuming that nonlinearity exists within a BB, the linkage identification by nonlinearity check (LINC) (Munetomo and Goldberg 1998) 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 is equal to the fitness difference caused by simultaneously perturbing the two variables, linearity is confirmed, and thus, these two variables are considered independent. Instead of non-linearity, the descendant of LINC, linkage identification by non-monotonicity detection (LIMD) (Munetomo and Goldberg 1999), adopts non-monotonicity to detect interactions among variables. Compared to EDAs, the low salience BBs are unlikely ignored in these approaches. However, since obtaining fitness differences requires extra function evaluations, perturbation methods are usually considered demanding more computational efforts to detect linkages. In addition to empirical studies Heckendorn and Wright (2004) generalised these methods through Walsh analysis to obtain theoretical resource requirements. Zhou, Sun, and Heckendorn (2007) and Zhou, Heckendorn, and Sun (2008) later extended this study from the binary domain to high-cardinality domains.

An interesting approach combining the ideas of EDAs and perturbation methods, called the dependency detection for distribution derived from fitness differences (D5), was developed by Tsuji, Munetomo, and Akama (2006). D5 detects the dependencies of variables by estimating the distributions of strings clustered according to fitness differences. For each variable, D5 calculates fitness differences by perturbations on that variable in 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 an algorithmic parameter for problem complexity, i.e. the number of variables in a linkage set. The determined k variables are considered forming a linkage set. D5 can detect dependencies for a class of functions that are difficult for EDAs, e.g. functions containing low salience BBs, and requires less computational cost than other perturbation methods do. However, its major constraint is that it relies on parameter k, which may not be available due to the limited information of the problem structure. As a side-effect to parameter k, D5 might be fragile in the situation where the problem is composed of subproblems of different sizes. Moreover, Ting, Zeng, and Lin (2010) recently utilised another data mining technique, Apriori Algorithm, to learn potential association rules between decision variables for linkage discovery. They reported that their proposal can improve D5 in terms of solution quality and efficiency.

In our previous work, we proposed inductive linkage identification (ILI) based on perturbations and the integration with the Interative Dichotomiser (ID3) (Quinlan 1986) algorithm, which is widely used in machine learning. ILI is an unsupervised method without any parameter for the complexity of BBs.
Its scalability and efficiency against the increasing problem sizes have been demonstrated (Chuang and Chen 2007, 2008; Huang and Chen 2009, 2010). Compared to the conventional perturbation methods, such as LINC and LIMD, ILI utilises a data mining technique to analyse objective functions. Compared to $D^5$, which uses clustering, and the method proposed by Ting et al. (2010), which uses Apriori algorithm, ILI adopts the ID3 algorithm and behaves quite differently. In this article, we aim to address more detailed characteristics of ILI in order to gain deeper insights and better understandings of linkage learning. In particular, problems constructed by non-overlapped BBs 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 BBs.

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

2. Linkage and BBs

In this section, we briefly review the definitions and terminologies which will be used through out this article. As stated by de Jong, Watson, and Thierens (2005), ‘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 will be examined on all possible settings of the other variables.

Although obtaining the full linkage information is computationally expensive, linkage should be estimated using a reasonable amount of efforts if the target problem is decomposable. According to the Schema theorem (Holland 1992), short, low-order and highly fit substrings increase their share to be combined. Also stated in the BB hypothesis, GAs implicitly decompose a problem into sub-problems by processing BBs. It is considered that combining small parts is important for GAs and is consistent with human innovation (Goldberg 2002). 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 $s$ of length $\ell$ be described as a series of variables, $s = s_1s_2 \ldots s_{\ell}$. We assume that $s = s_1s_2 \ldots s_{\ell}$ is a permutation of the decision variables $x = x_1x_2 \ldots x_{\ell}$ to represent the encoding scheme adopted by GA users. The fitness of string $s$ is then defined as

$$f(s) = \sum_{i=1}^{m} f_i(s_i),$$  \hspace{1cm} (1)

where $m$ is the number of sub-functions, $f_i$ is the $i$-th sub-function and $s_i$ is the substring to $f_i$. Each $v_i$ is a vector specifying the substring $s_i$. For example, if $v_i = (1, 2, 4, 8)$, $s_i = s_1s_2s_4s_8$. 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 of $v_i$. 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, BBs, is referred to as the candidate solutions to sub-function $f_i$. In this article, 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 ILI will be presented. Then, the ID3 algorithm, which is proposed and widely utilised in the field of machine learning, will be 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:

$$f(s_1s_2 \ldots s_{\ell}) = f_1(s_1s_2s_3s_4s_5) + f_2(s_6s_7s_8)$$

$$d_{f_1}(s) = f(s_1s_2 \ldots s_{\ell}) - f(s_1s_2s_3s_4s_5)$$

$$= (f_1(s_1s_2s_3s_4s_5) + f_2(s_6s_7s_8)) - (f_1(s_1s_2s_3s_4s_5) + f_2(s_6s_7s_8))$$

$$= f_2(s_6s_7s_8) - f_1(s_1s_2s_3s_4s_5).$$  \hspace{1cm} (2)

Equation (2) indicates that the fitness difference $d_{f_1}$ should be affected only by the bits belonging to the same sub-functions as the perturbed bits $s_1$, which are $s_1s_2s_4s_5$. 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: given a condition composed of attributes, an agent (algorithm) should learn to make a decision with the given training instances. When the decision-making method is adopted for conducting linkage learning, decision variables are regarded as attributes and the fitness difference values stand for class labels. With this simple and direct mapping, linkage learning in GAs can potentially be handled with certain well-developed methods in machine learning.

3.1. Decision tree learning: ID3

The ID3 algorithm was proposed by Quinlan (1986) for the purpose of constructing a decision tree on a set of training instances. In its basic form, ID3 constructs a decision tree in a top–down manner without backtracking. When a decision tree is being constructed, each attribute is evaluated using a statistical property, to measure how well the attribute alone classifies the training instances. The best attribute, which leads to the highest information gain, 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 the selected attribute, and the training instances are split into appropriate descendant branches. The entire process is repeated on the training instances associated with each descendant node.

The statistical property, information gain, of each attribute is simply the expected reduction in the impurity of instances after classifying the instances with the selected attribute. The impurity of an arbitrary collection of instances is 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

$$\text{Entropy}(D) \equiv -\sum_{i=1}^{c} -p_i \log_2 p_i,$$ (3)

where $p_i$ is the proportion of $D$ belonging to class $i$. For simplicity, in all the calculations involving entropy, we define $0 \log_0 0$ to be 0. In terms of entropy, the information gain, $Gain(D, A)$, of an attribute $A$ relative to a collection of instances $D$, is defined as

$$Gain(D, A) \equiv \text{Entropy}(D) - \sum_{v \in \text{Val}(A)} \frac{|D_v|}{|D|} \text{Entropy}(D_v),$$ (4)

where $\text{Val}(A)$ is the set of all possible values for attribute $A$ and $D_v$ is the subset of $D$ of which attribute $A$ has value $v$. In summary, ID3 can be described as the pseudo code given in Algorithm 1.

**Algorithm 1: Pseudo code of ID3**

```
procedure ID3(D)
    Stop if no further classification is need
    for each attribute $A$
do
        Calculate $Gain(D, A)$
    end for
    Select the attribute with the highest information gain as a tree node
    for each possible value $v$ of the selected attribute
        Create a branch for $D_v$, the subset of $D$ of which the selected attribute has value $v$
        Call ID3($D_v$) to construct this subtree
    end for
end procedure
```

In the proposal of ILI (Chuang and Chen 2007), the ID3 algorithm is adopted as a classification and relationship extraction mechanism. Linkage learning is then achieved by a sequence of decision tree constructions. In a classification problem, a training instance is composed of a list of attributes describing the instance and a target value which the decision tree is supposed to predict after training. For the purpose of linkage identification, the list of attributes is the solution string, and the target value is the fitness difference caused by perturbations.

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:

$$f_{trap_k}(s_1s_2 \cdots s_k) = trap_k\left(u = \sum_{i=1}^{k} s_i\right)$$

$$= \begin{cases} k, & \text{if } u = k; \\ k - 1 - u, & \text{otherwise}, \end{cases}$$ (5)

where $u$ is the number of ones in the string $s_1s_2 \cdots s_k$. Suppose that we are dealing with an eight-bit problem

$$f(s_1s_2 \cdots s_8) = f_{trap_3}(s_1s_2s_3s_4s_5) + f_{trap_3}(s_6s_7s_8),$$ (6)

where $s_1s_2 \cdots s_8$ is a solution string. In the black-box optimisation scenario, the structural decomposition of the objective function is unknown. Our goal here is to
identify the two linkage sets \( V_1 = \{1, 2, 3, 4, 5\} \) and \( V_2 = \{6, 7, 8\} \), which correspond to the problem structural decomposition.

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 interdependent on \( s_1 \). Note that the choice of first operating on \( s_1 \) in this example is not mandatory. Any un-grouped decision variable in the encoding may be chosen as the root node. The third column of Table 1 records the fitness differences, \( d_{f_1} \), 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 \( s_1s_2\cdots s_8 \) are regarded as attributes of the instances, and the fitness differences \( d_{f_1} \) are the target values/class labels. Corresponding to Table 1, an ID3 decision tree shown 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 shown in Table 2. Conducting the same procedure, an ID3 decision tree presented in Figure 2 is obtained. By gathering all the decision variables used in 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 BBs without assumptions. Such an ability implies that ILI should be capable of finding all relations among these variables as long as the

<table>
<thead>
<tr>
<th>( s_1s_2\cdots s_8 )</th>
<th>( f )</th>
<th>( d_{f_1} )</th>
<th>( s_1s_2\cdots s_8 )</th>
<th>( f )</th>
<th>( d_{f_1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001 011</td>
<td>3</td>
<td>1</td>
<td>T0010 110</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>0011 111</td>
<td>5</td>
<td>1</td>
<td>T0011 000</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>0010 001</td>
<td>3</td>
<td>1</td>
<td>T0101 010</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>0010 100</td>
<td>3</td>
<td>1</td>
<td>T0101 100</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>0010 111</td>
<td>5</td>
<td>1</td>
<td>T0110 110</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>0110 111</td>
<td>5</td>
<td>1</td>
<td>T1010 110</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>0110 111</td>
<td>6</td>
<td>1</td>
<td>T1011 100</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>0110 111</td>
<td>2</td>
<td>1</td>
<td>T1001 011</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>0110 110</td>
<td>1</td>
<td>1</td>
<td>T1001 111</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>0111 001</td>
<td>1</td>
<td>1</td>
<td>T1010 011</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>0111 110</td>
<td>1</td>
<td>1</td>
<td>T1111 011</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>0111 111</td>
<td>3</td>
<td>-5</td>
<td>T1111 000</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>0100 010</td>
<td>3</td>
<td>-1</td>
<td>T1100 101</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>0001 010</td>
<td>2</td>
<td>-1</td>
<td>T1111 010</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0010 101</td>
<td>2</td>
<td>-1</td>
<td>T1111 111</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 1. ID3 decision tree constructed according to Table 1.

<table>
<thead>
<tr>
<th>( s_1s_2\cdots s_8 )</th>
<th>( f )</th>
<th>( d_{f_6} )</th>
<th>( s_1s_2\cdots s_8 )</th>
<th>( f )</th>
<th>( d_{f_6} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>11100 000</td>
<td>1</td>
<td>0</td>
<td>10101 100</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10011 000</td>
<td>1</td>
<td>0</td>
<td>01101 100</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11011 001</td>
<td>0</td>
<td>0</td>
<td>00100 100</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>01111 001</td>
<td>0</td>
<td>0</td>
<td>00010 101</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>00100 001</td>
<td>3</td>
<td>0</td>
<td>10110 011</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11111 010</td>
<td>5</td>
<td>0</td>
<td>11110 101</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10101 110</td>
<td>1</td>
<td>0</td>
<td>01101 101</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11100 110</td>
<td>1</td>
<td>0</td>
<td>01110 101</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10001 110</td>
<td>2</td>
<td>0</td>
<td>01111 101</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11011 110</td>
<td>0</td>
<td>0</td>
<td>01110 110</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10000 110</td>
<td>3</td>
<td>0</td>
<td>10101 110</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>01101 110</td>
<td>1</td>
<td>0</td>
<td>01111 110</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>00001 111</td>
<td>3</td>
<td>-3</td>
<td>10010 110</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>00011 111</td>
<td>3</td>
<td>-3</td>
<td>00111 111</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>11010 111</td>
<td>1</td>
<td>-3</td>
<td>00011 111</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>11001 111</td>
<td>1</td>
<td>-3</td>
<td>01000 111</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>11111 111</td>
<td>5</td>
<td>-3</td>
<td>00111 111</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>11100 111</td>
<td>1</td>
<td>-3</td>
<td>11001 111</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>01010 111</td>
<td>2</td>
<td>-3</td>
<td>00110 111</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>10111 100</td>
<td>0</td>
<td>0</td>
<td>01111 111</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2. Population perturbed at \( s_6 \).
population size is sufficiently large to provide significant statistics.

3.3. Inductive linkage identification

In this section, the idea demonstrated in the previous section is formalized as an algorithm, which is called ILI. The pseudo code of ILI is presented in Algorithm 2. Conceptually, ILI includes the following three main steps:

1. Calculate fitness differences by perturbations;
2. Construct an ID3 decision tree;
3. Consider the tree nodes as a linkage set.

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

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

Algorithm 2: Inductive linkage identification

```plaintext
procedure IDENTIFYLINKAGE(f, l)
    Initialise 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 ← V - {v}
        for each string s = s₁, s₂, ..., sₙ in P do
            Perturb sᵣ.
            dfᵣ ← fitness difference caused by perturbation.
        end for
        Construct an ID3 decision tree using (P, df).
        for each decision variable sᵣ in tree do
            Vᵣ ← Vᵣ UNION {j}
            V ← V - {j}
        end for
    end while
    return linkage sets V₁, V₂, ..., Vₘ
end procedure
```

As clearly shown in Algorithm 2, there is no parameter needed for indicating the complexity of sub-functions. That is, ILI does not rely on any assumption on the size of BBs while other existing perturbation methods usually require the maximum size of BBs to be specified. This property distinguishes ILI from 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 decision tree construction.

From our previous studies (Chuang and Chen 2007, 2008), we know that the required population size grows linearly with the problem size while the BBs size is constant. Such results indicate that ILI is more efficient than LINC, O(ℓ²) = O(k²m²) (Munetomo and Goldberg 1998), and similar to D⁵, O(l) = O(km) (Tsuji et al. 2006), where ℓ is the problem size, k is the size (i.e. length or order) of BBs, and m is the number of BBs. Note that the comparison focuses on the amount of required computational resource instead of the identification quality. This is because given sufficient computational resource, all these methods can successfully identify every BB. In order to gain further understandings on the flexibility and applicability of ILI, in the next section, experiments on the BBs of different sub-functions as well as lengths are conducted and discussed.

4. Experiments and results

Experiments and results of ILI on binary and non-overlapped ADFs will be presented in this section. These experiments are designed to gain a better understanding of the behaviour of ILI on problems of different sub-functions compositions, including size-varied, size-mixed BBs and different sub-functions.

The required population size reflects the behaviour of ILI. Therefore, our experiments are designed to
obtain the minimal population sizes required for different problem configurations. For a given problem, first a population size assuring successful trials of linkage identification, which means correctly identifying all the BBs within the problem for 30 consecutive and independent runs, is obtained by doubling the population size from 2500 until the first successful trial is archived. Once the upper bound of population sizes $P_U$ is found, the required population size is determined in a bisection manner: the population size $P = (P_L + P_U)/2$ will be configured for ILI, where $P_L = 1$ for the first iteration. If ILI can succeed with this population size $P$, then $P$ 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 smaller than a predefined distance, which is 2 in this study, and the last tested population size is considered as the minimal requirement for the current problem.

4.1. Different BB sizes

This section describes the experiment on problems of identical overall sizes but with different-sized subfunctions. From our experimental results with different configurations of the BB size $k$ and the number of BBs $m$, we group those results with the overall problem sizes and arrange them with the BB size $k$. Thus, the results of the same problem size with different $k$ can be examined.

Figure 3(a) and (b) shows 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 straight lines indicate that for identical overall problem sizes, the requirements of both the population size and the function evaluation grow 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 0.8 and $C$ varies with different problem sizes. Earlier studies by Munetomo and Goldberg (1998) and Heckendorn and Wright (2004), respectively, suggested an empirical and a theoretical upper bounds of function evaluations, which are both in the form of $2^k \ell \log(\cdot)$ for problems of $\ell$ bits, composed of order-$k$ BBs and each BB sharing $j$ bits with others. Reviewing our empirical results with the upper bounds, ILI shows the same computational complexity of the exponential growth with $k$ when overall sizes remain constant, such an observation is consistent with the upper bounds reported in the literature. However, the regression gives 0.8 as the base of exponent and thus indicates a practically better efficiency compared to the suggested upper bound when the complexity of sub-problem increases.

4.2. Mixed BB 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 BBs of order-$k_1$ and order-$k_2$ trap functions as

$$\text{trap}_{k_1+k_2}(\cdot) = \sum_{i=1}^{m} (\text{trap}_{k_1}(\cdot) + \text{trap}_{k_2}(\cdot)), \quad (7)$$

where $m$ is the number of $\text{trap}_{k_1}$ and $\text{trap}_{k_2}$. By designing the experiments in this way, the empirical results can 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 $\text{trap}_{k_1+k_2}(\cdot)$, two
results of the same amount of \( \text{trap}_{k_1} \) and \( \text{trap}_{k_2} \) from experiments in Section 4.1 are summed up to get the same problem size and total number of BBs, interpolation is utilised when there are no results of such configurations. These calculated numbers are denoted as \( \text{trap}_{k_1} + \text{trap}_{k_2} \) in Figure 4 with the experimental results \( \text{trap}_{k_1+k_2} \).

First, these results show that ILI is capable of detecting BBs of different sizes within one problem without any extra information regarding the complexity of sub-problems. Second, comparing with calculated data, it can be seen that although ILI requires more function evaluations for the problems composed of mixed BB sizes, the growth rate is still linear or very close to linear. The observation indicates that identifying size-varied BBs 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. BBs of various elementary functions

Despite of using \( \text{trap}_k \) functions as the sub-function to construct BBs, the capability of ILI to handle BBs formed by other functions shown 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 used in this section.

Figure 6 shows the experimental results. The required population sizes and function evaluations of \( \text{trap}_4, \text{nith}_4, \text{tmmp}_4 \) and \( \text{valley}_4 \) are plotted together, and the standard deviation of the results for \( \text{trap}_4 \) is also shown in the figures. Because the population and function evaluation requirements of these problems are similar, the behaviour 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 variables as long as a sufficiently large population is employed to provide significant statistics.

### 5. Summary and conclusions

In this article, we examined ILI on several different configurations of BBs in order to gain better understandings. We focused on the mixed sizes of BBs 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 mixed sizes of BBs, and the applicability of ILI on BBs formed with various elementary functions.

From the experiments of BB sizes, it is demonstrated that the required function evaluations grow exponentially with the size of BBs 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-O while ILI demands less computational resource in practice. On the other hand, if computationally expensive real-world problems, such as parametric engineering design (Saridakis and Dentsoras 2009), are handled, and the optimisation framework has to be made much more efficient, techniques of the surrogate-assisted
evolutionary algorithm (SAEA) (Sastry, Goldberg, and Pelikan 2001; Jin 2003; Lim, Jin, Ong, and Sendhoff 2010a; Lim, Ong, Setiawan, and Idris 2010b) may be adopted and utilised.

Another observation is that when ILI performs on problems composed of mixed-sized BBs, 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 BBs are quite similar. Thus, this series of experiments evidentially proves that ILI behaves similarly when handling sub-problem of different types.

Figure 5. Elementary functions adopted in the series of experiments in Section 4.3; (a) \(trap_4\), (b) \(nith_4\), (c) \(tmmp_4\) and (d) \(valley_4\).

Figure 6. Experimental results on different 4-bits BB types: (a) required population sizes and (b) required function evaluations.
As a consequence, we can now know that the most important factor that affects ILI’s ability to identify linkage is the size of BBs. Compared with the BB size, ILI is relatively insensitive to other factors commonly studied by the related work, including the overall problem size, the number of BBs, and the type of BBs. Hence, ILI can be considered as a good linkage learning technique and can be adopted as a tool for analysing structures of target problems or a pre-processing procedure in frameworks of GAs.

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 GA framework, handling real-world applications with ILI, exploring ILI’s capability of analysing problem structures and understanding the nature of linkage learning via getting deeper insights of ILI. As for the immediate future studies, the idea of ‘linkage identification as decision learning’ can be adapted to work with other advanced decision tree techniques. Characteristics of different decision tree algorithms might exhibit behaviour of different kinds and give us a better understanding of linkage identification. Such knowledge can be utilised to practically help the algorithmic development of GAs and theoretically reveal the working principle of evolutionary computation.

Acknowledgements
The work was supported in part by the National Science Council of Taiwan under Grant NSC 99-2221-E-009-123-MY2. The authors are grateful to the National Center for High-performance Computing for computer time and facilities.

Notes on contributors

Ying-ping Chen received his BS and MS degrees in Computer Science and Information Engineering from National Taiwan University, Taiwan, in 1995 and 1997, respectively, and PhD in 2004 from the Department of Computer Science, University of Illinois at Urbana-Champaign, Illinois, USA. He has been an Assistant Professor from 2004 to 2009 and an Associate Professor since 2009 in the Department of Computer Science, National Chiao Tung University, Taiwan. His research interests in the field of genetic and evolutionary computation include theories, working principles, particle swarm optimisation, estimation of distribution algorithms, linkage learning techniques and dimensional/facet-wise models.

Chung-Yao Chuang received his BS and MS degrees in Computer Science from National Chiao Tung University, Taiwan, in 2006 and 2008, respectively. He is currently working at Academia Sinica, Taiwan for obligated citizen service and looking forward to being included in a PhD program starting at Fall 2012. His research interests include linkage problem in evolutionary algorithms, estimation of distribution algorithms, evolutionary computation and machine learning in general.

Yu-an-Wei Huang received the BS and MS degrees in Computer Science from National Chiao Tung University, Taiwan, in 2008 and 2010, respectively. His research interests include linkage learning techniques, machine learning, and artificial intelligence.

References


