Linkage Identification by Non-monotonicity Detection for Overlapping Functions

Masaharu Munetomo
Graduate School of Engineering
Hokkaido University
North 13, West 8, Kita-ku,
Sapporo 060–8628, Japan
munetomo@eng.hokudai.ac.jp

David E. Goldberg
Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
104 South Mathews Avenue
Urbana, IL 61801, USA
deg@uiuc.edu

Abstract

This paper presents the linkage identification by non-monotonicity detection (LIMD) procedure and its extension for overlapping functions by introducing the tightness detection (TD) procedure. The LIMD identifies linkage groups directly by performing order-2 simultaneous perturbations on a pair of loci to detect monotonicity/non-monotonicity of fitness changes. The LIMD can identify linkage groups with at most order of k when it is applied to $O(2^k)$ strings. The TD procedure calculates tightness of linkage between a pair of loci based on the linkage groups obtained by the LIMD. By removing loci with weak tightness from linkage groups, correct linkage groups are obtained for overlapping functions, which were considered difficult for linkage identification procedures.

Keywords

Linkage identification, monotonicity detection, population sizing, overlapping functions.

1 Introduction

The power of genetic search lies in its processing of building blocks (BBs)—essential subcomponents of solutions—through crossover and selection. Recent work has shown that effective BB mixing is absolutely essential. For the effective mixing, a set of loci that belongs to a BB needs to be tightly linked in crossover to avoid disruptions. The tightness of loci is referred to as *linkage*, and a set of loci tightly linked is called a *linkage set* or a *linkage group*. In genetics, linkage is "the tendency for alleles of different genes to be passed together from one generation to the next" (Winter et al., 1998). This definition indicates that such genes are mapped closely in the same chromosome. In genetic algorithm (GA) literature, this indication does not seem useful because we do not want to detect linkage groups found in the encoded strings, which is completely trivial, but want to detect linkage groups for the underlying structure of the problem which is also dependent upon the encoding system employed. For some GA-easy problems, we can encode strings to ensure tight linkage, however, we cannot take such a simple approach for all problems. For problems where we cannot ensure tight linkage in advance, it is necessary to identify linkage groups. Once correct linkage groups are identified, it becomes easy for GAs to mix BBs effectively without disrupting them.

To identify linkage groups, several algorithms were proposed. They are classified roughly into the following three categories:

1. Direct detection of bias in probability distribution
2. Direct detection of fitness changes by perturbations
3. Indirect detection along genetic search of BBs

For the first category, several algorithms such as the estimation of distribution algorithm (EDA) (Mühlenbein and Paaß, 1996), the univariate marginal distribution algorithm (UMDA) (Mühlenbein, 1997), the bivariate marginal distribution algorithm (BMDA) (Pelikan and Mühlenbein, 1999), and the Bayesian optimization algorithm (BOA) (Pelikan et al., 1998) were proposed to identify linkage groups by detecting bias on probability distributions after selections. For the second category, the gene-expression messy-GA (GEMGA) (Kargupta, 1996c) calculates the change of fitness values in each locus of each string by performing perturbations to detect loci whose alleles may form local optima. To collect such loci among strings in a population, the algorithm can identify possible BBs for the problem. The revised GEMGA (Kargupta, 1996a; Kargupta et al., 1997) introduces an order-2 simultaneous perturbation method to detect linkage in addition to the order-1 perturbation to detect local optima. The order-2 perturbations detect invariance of the change in one position by a perturbation of the other's, which is considered to detect a linearity to be included in a linkage set. The GEMGA in Bandyopadhyay et al. (1998) does not employ the previous approach to detect linkage. Instead, it also considers the value of locus and collects loci whose alleles form local optima caused by perturbations of the same direction as a linkage group. This is simply because when a locus is considered as a member of loci whose alleles form a local optimum, it is not necessary to be a member in a global optimum. By checking the value of the locus itself, before a perturbation for different strings (contexts), the algorithm can increase the reliability of detecting the loci whose alleles form a global optimum.

For the third category, the linkage learning GA (LLGA) (Harik, 1997) employs a two-point like crossover over circular strings to grow tight linkages of BBs. The LLGA works effectively on exponentially-scaled problems, which are the sum of exponentially weighted subfunctions, but fails to exploit linkage groups in uniformly-scaled problems. This is because simultaneous search for linkage groups and BBs may cause a negative feedback effect that prevents obtaining correct results. In this category, another method based on the idea of a "selfish gene" was also proposed (Corno et al., 1998).

In the following, we concentrate our discussion on the second category of linkage identification. The linkage identification by nonlinearity check (LINC) procedure (Munetomo and Goldberg, 1998) was proposed to identify linkage groups directly by employing a bitwise perturbation technique that was pioneered by Kargupta in his revised GEMGA (Kargupta, 1996b). The LINC did the opposite of the GEMGA: the LINC detects nonlinearity for a pair of loci to be included in a linkage group, while the revised GEMGA detects invariance of changes equaling linearity to be identified as a linkage. Unlike the GEMGA, the LINC does not rely on the local/global optimality of the problem; it only considers whether the problem can be decomposed into smaller subproblems or not. Once a problem is divided into subproblems based on the obtained linkage groups, it becomes easy for GAs to concentrate on testing and mixing BBs. The LINC can identify correct linkage groups for order-k delineable problems by using order-2 perturbations applied to $O(2^k)$ strings. In addition, to consider GA-easy nonlinearity, a condition of *allowable nonlinearity* (Munetomo and Goldberg, 1998) was introduced to relax the LINC condition, which unlinks previously detected linkage groups to be more accurate for not only quasi-linearly separable functions

but also GA-easy nonlinear functions of BBs.

In this paper, we propose the linkage identification by non-monotonicity detection (LIMD) procedure which detects linkage groups by performing perturbations between a pair of loci for all the strings in a population. This procedure enables us to detect accurate linkage groups for GA-easy nonlinear functions of BBs. We discuss equality between the LIMD and the LINC with allowable nonlinearity (LINC-AN). We also design a tightness detection (TD) procedure that detects meta-level tightness existing in the linkage sets obtained by the LIMD. The LIMD with TD procedure (LIMD-TD) is expected to identify linkage groups correctly for functions that have overlapping coefficients among their subfunctions.

This paper continues as follows: first, we introduce the linkage identification by nonlinearity check (LINC) procedure which checks any nonlinearity to detect linkage groups. Second, we discuss a class of easy nonlinearity for GAs as an allowable nonlinearity for the LINC. Third, we present the LIMD condition and discuss equality between the LIMD and the LINC with AN. Then, we present the TD procedure for the LIMD. We estimate the population size necessary for the identification and also consider the size for noisy functions. Finally, we perform experiments on non-overlapping and overlapping test functions to validate its ability to detect linkage groups.

2 Nonlinearity Check

The linkage identification by nonlinearity check (LINC) procedure identifies linkage groups by detecting nonlinearity caused by perturbations. If an arbitrary nonlinearity is detected by perturbations in a pair of loci for at least one string in a population, they are included in a linkage group. This is based on an assumption that nonlinearity must be existent within loci to form a BB; otherwise, they are separable to lower order BBs.

In the following, we consider a string $s = s_1 s_2 s_3 \cdots s_l$ and define changes of fitness values by bit-wise perturbations to s as follows:

$$\Delta f_i(s) = f(..\bar{s}_i.....) - f(..s_i.....) \quad (1)$$

$$\Delta f_j(s) = f(.....\bar{s}_j..) - f(.....s_j..) \quad (2)$$

$$\Delta f_{ij}(s) = f(..\bar{s}_i.\bar{s}_j..) - f(..s_i.s_j...), \quad (3)$$

where $\bar{s}_i = 1 - s_i$ and $\bar{s}_j = 1 - s_j$ in binary strings.

If $\Delta f_{ij}(s) = \Delta f_i(s) + \Delta f_j(s)$, that is, changes of fitness values by perturbations on s_i and s_j are additive, it indicates a linear interaction between them. If $\Delta f_{ij}(s) \neq \Delta f_i(s) + \Delta f_j(s)$, they are not additive, and it indicates nonlinearity.

Checking nonlinearity in one string is not enough, because there may exist linearity inside a BB in some contexts (for example, a trap function is linear along its deceptive attractor). Therefore, all strings in a properly sized population must be checked. If linearity is detected for all the strings in a pair of loci, it is safe for them to be unlinked.

To store linkage groups, we assign a linkage set (a list of loci which are tightly linked) to each locus. Unlike the GEMGA which assigns a linkage set to each string, the linkage set of the LINC stores linkage information for all the strings in a population. To obtain linkage sets, the following procedure is performed on each pair of loci (i, j) for each string s in a population.

1. If $\Delta f_{ij}(s) \neq \Delta f_i(s) + \Delta f_j(s)$, then s_i and s_j are members of a linkage set, so we add i to the linkage set of locus j and j to the linkage set of locus i.

2. If $\Delta f_{ij}(s) = \Delta f_i(s) + \Delta f_j(s)$, then s_i and s_j may not be members of a linkage set, or they are linked but linearity exists in the current context. We do nothing in this case.

We can introduce the value ϵ that specifies the amount of error allowed for linearity/nonlinearity detection and replace the above conditions by $(|\Delta f_{ij} - (\Delta f_i + \Delta f_j)| > \epsilon)$ and $(|\Delta f_{ij} - (\Delta f_i + \Delta f_j)| \leq \epsilon)$. If the problem is completely decomposable to nonoverlapping subproblems without noise, then we can set the value of ϵ at zero. If the fitness function is only quasi-decomposable or noisy, then we need to set ϵ at a positive value depending upon the problem. A similar condition was proposed in the definition of conjugate schemata (Kazadi, 1997) to find the transformation of encoding that reduces complexity. However, the definition only considers local linearity/nonlinearity in function domain and does not propose sampling procedures on encoded strings.

3 Allowable Nonlinearity

If a problem is linearly decomposable, checking only arbitrary nonlinearity is enough to yield correct linkage sets. In general, it is not enough because fitness changes by perturbations in a pair of loci need not be exactly additive in order for them to be GA-easy. When we detect nonlinearity with a reinforcing contribution to fitness changes by simultaneous perturbations in a pair of loci, GAs can improve fitness values by combining the offspring obtained by the perturbations. Therefore, only checking nonlinearity may produce overspecified linkage sets from GA's mixing point of view.

Consequently, it is also necessary to detect easy nonlinearity for GAs to be excluded from linkage sets. In the following, we present an "allowable" nonlinearity for a GA-easy nonlinearity. When $\Delta f_i(s) > 0$ and $\Delta f_j(s) > 0$, we expect fitness improvements on successive perturbations in s_i and s_j. If the overall effect of the successive perturbations on fitness value is additive, i.e., if we have $\Delta f_{ij}(s) = \Delta f_i(s) + \Delta f_j(s)$, then s_i and s_j are decomposable and the GA can improve fitness values by combining perturbations in the loci. Even when we do not have such an additive property, the GA can improve fitness value. This happens when the following condition is satisfied:

$$\Delta f_{ij}(s) > \Delta f_i(s) \text{ and } \Delta f_{ij}(s) > \Delta f_j(s) \tag{4}$$

When we set $f_i(s) = f(s) + \Delta f_i(s)$, $f_j(s) = f(s) + \Delta f_j(s)$, and $f_{ij}(s) = f(s) + \Delta f_{ij}(s)$, the condition above is identical to $(f_{ij}(s) > f_i(s)$ and $f_{ij}(s) > f_j(s))$, which means that successive perturbations in s_i and s_j cause monotone increases of fitness values along $f(s) \rightarrow f_i(s) \rightarrow f_{ij}(s)$ and $f(s) \rightarrow f_j(s) \rightarrow f_{ij}(s)$. Problems that satisfy the condition are GA-easy in the loci (i, j) because positive changes of $\Delta f_i(s)$ or $\Delta f_j(s)$ will increase the number of strings through selection, and the combination of the changes will also improve their fitness values. Therefore, we do not need to include them in the linkage set. The case of negative changes when, $(\Delta f_i(s) < 0$ and $\Delta f_j(s) < 0)$ becomes identical to those in the positive case when we consider it on all possible contexts.

It should be noted that we need to check whether the above condition is satisfied in all possible substrings (or almost all; we can relax the condition, but it may cause a problem in nonlinearity detection) for each linkage set detected by the nonlinearity check. That is, to

remove a pair from the linkage set, the above relation needs to be satisfied in all contexts that satisfy $\Delta f_i(s) > 0$ and $\Delta f_j(s) > 0$. Population sizing is discussed in Section 7.

4 Non-monotonicity Detection

Instead of checking nonlinearity in the LINC procedure, the linkage identification by non-monotonicity detection (LIMD) procedure we propose in this paper checks violation of monotonicity conditions to detect linkage groups. The procedure adds a pair of loci (i, j) to the linkage set when the following condition is *not* satisfied in at least one string in a population.

$$\begin{aligned}&\text{if } (\Delta f_i(s) > 0 \text{ and } \Delta f_j(s) > 0)\\ &\quad \text{then } (\Delta f_{ij}(s) > \Delta f_i(s) \text{ and } \Delta f_{ij}(s) > \Delta f_j(s)) \qquad (5)\\ &\text{if } (\Delta f_i(s) < 0 \text{ and } \Delta f_j(s) < 0)\\ &\quad \text{then } (\Delta f_{ij}(s) < \Delta f_i(s) \text{ and } \Delta f_{ij}(s) < \Delta f_j(s)) \qquad (6)\end{aligned}$$

In the above equation, $\Delta f_i(s)$, $\Delta f_j(s)$, and $f_{ij}(s)$ are the same as in the LINC conditions. When we also define $f_i(s)$, $f_j(s)$, and $f_{ij}(s)$ to be the same as in the previous discussion on the LINC, we can rewrite the above conditions as follows:

$$\begin{aligned}&\text{if } (f_i(s) > f(s) \text{ and } f_j(s) > f(s))\\ &\qquad \text{then } (f_{ij}(s) > f_i(s) \text{ and } f_{ij}(s) > f_j(s)) \qquad (7)\\ &\text{if } (f_i(s) < f(s) \text{ and } f_j(s) < f(s))\\ &\qquad \text{then } (f_{ij}(s) < f_i(s) \text{ and } f_{ij}(s) < f_j(s)) \qquad (8)\end{aligned}$$

These indicate either monotone increases ($f(s) < f_i(s) < f_{ij}(s)$, $f(s) < f_j(s) < f_{ij}(s)$) or decreases ($f(s) > f_i(s) > f_{ij}(s)$, $f(s) > f_j(s) > f_{ij}(s)$) of fitness values by a series of perturbations at loci i and j.

A pseudo-C code of the LIMD procedure is shown in Appendix A. The procedure is applied to a population of randomly initialized binary strings (we omit the initialization of strings in the code because it is trivial), each of which is checked by the conditions (5) and (6) in each pair of loci. The monotonicity condition for the negative case (when $\Delta f_i(s) < 0$ and $\Delta f_j(s) < 0$) becomes essentially the same as that for the positive case when we consider all the possible strings. However, to reduce the number of strings necessary to detect linkage, we also check the negative case. In the procedure, for each string s: first, a perturbation in position i is applied to calculate `df1`$= \Delta f_i(s)$; second, a perturbation in position j is applied to have s' and calculate `df2`$= \Delta f_j(s)$, and then another perturbation in position i is applied to s' to calculate `df12`$= \Delta f_{ij}(s)$; third, employing the calculated fitness differences by perturbations, the algorithm checks whether the LIMD condition is satisfied or not. If any violation of the monotonicity condition is detected, the pair of loci (i, j) are included in their linkage sets, that is, locus i is included in the `linkage_set[j]` and locus j is included in the `linkage_set[i]`.

5 Equality Between the LINC-AN and the LIMD

In this section, we discuss equality between the conditions of the LIMD and the LINC with allowable nonlinearity. We can prove that the above condition for monotonicity detection is the same as that of the LINC with allowable nonlinearity (LINC-AN).

We list the LINC, the LINC-AN, and the LIMD conditions as follows. These are the conditions under which a pair of loci (i, j) should be included in the linkage sets.

LINC: $\exists s(\Delta f_{ij}(s) \neq \Delta f_i(s) + \Delta f_j(s))$

LINC-AN: $\exists s(\Delta f_{ij}(s) \neq \Delta f_i(s) + \Delta f_j(s))$ and $\neg(\forall s($ if $\Delta f_{ij}(s) \neq \Delta f_i(s) + \Delta f_j(s)$ and $(\Delta f_i(s) > 0$ and $\Delta f_j(s) > 0)$ then $(\Delta f_{ij}(s) > \Delta f_i(s)$ and $\Delta f_{ij}(s) > \Delta f_j(s))))$

LIMD: $\exists s\neg($ if $(\Delta f_i(s) > 0$ and $\Delta f_j(s) > 0)$ then $(\Delta f_{ij}(s) > \Delta f_i(s)$ and $\Delta f_{ij}(s) > \Delta f_j(s))))$

In the above conditions, $\Delta f_i(s)$ is the amount of change of fitness value by a perturbation of string s at locus i, $\Delta f_j(s)$ is that by a perturbation at locus j, and $\Delta f_{ij}(s)$ is that by simultaneous perturbations at loci i and j.

For simplicity, we define the following predicates,

- $E_{ij}(s) = \{\ \Delta f_{ij}(s) = \Delta f_i(s) + \Delta f_j(s)\ \}$
- $P_{ij}(s) = \{\ \Delta f_i(s) > 0$ and $\Delta f_j(s) > 0\ \}$
- $M_{ij}(s) = \{\ \Delta f_{ij}(s) > \Delta f_i(s)$ and $\Delta f_{ij}(s) > \Delta f_j(s)\ \}$

and we rewrite the conditions as follows:

LINC: $\exists s(\neg E_{ij}(s))$

LINC-AN: $\exists s(\neg E_{ij}(s))$ and $\neg(\forall s($ if $(\neg E_{ij}(s)$ and $P_{ij}(s))$ then $M_{ij}(s)))$

LIMD: $\exists s\neg($ if $P_{ij}(s)$ then $M_{ij}(s))$

When we replace **and** by $\wedge$, **or** by $\vee$, and **(if a then b)** by **(**$\neg$**a**$\vee$**b)**, we have the following conditions:

LINC: $\exists s(\neg E_{ij}(s))$

LINC-AN: $\exists s(\neg E_{ij}(s)) \wedge \neg(\forall s(\neg(\neg E_{ij}(s) \wedge P_{ij}(s)) \vee M_{ij}(s)))$

LIMD: $\exists s\neg(\neg P_{ij}(s) \vee M_{ij}(s))$

The conditions of the LINC-AN and the LIMD can be reduced as follows:

LINC-AN: $\exists s(\neg E_{ij}(s) \wedge P_{ij}(s) \wedge \neg M_{ij}(s))$

LIMD: $\exists s(P_{ij}(s) \wedge \neg M_{ij}(s))$

Here, we consider relations among $E_{ij}(s)$, $P_{ij}(s)$, and $M_{ij}(s)$. A predicate

$$\forall s(\text{ if } E_{ij}(s) \text{ and } P_{ij}(s) \text{ then } M_{ij}(s)) \tag{9}$$

is true because, if we have $\Delta f_{ij}(s) = \Delta f_i(s) + \Delta f_j(s)$ and $\Delta f_i(s) > 0$ and $\Delta f_j(s) > 0$, then we directly have $\Delta f_{ij}(s) = \Delta f_i(s) + \Delta f_j(s) > \Delta f_i(s)$ and $\Delta f_{ij}(s) = \Delta f_i(s) + \Delta f_j(s) > \Delta f_j(s)$. Therefore, by rewriting the condition, we know that $\forall s(\neg(E_{ij}(s) \wedge P_{ij}(s)) \vee M_{ij}(s))$ is true. By calculating the negation of this condition, we know that

$$\exists s(E_{ij}(s) \wedge P_{ij}(s) \wedge \neg M_{ij}(s)) \tag{10}$$

is false. Therefore, when we rewrite the LIMD conditions as in the following:

$$\exists s((E_{ij}(s) \vee \neg E_{ij}(s)) \wedge (P_{ij}(s) \wedge \neg M_{ij}(s))) \tag{11}$$

we know that the condition (11) becomes the same as that of the LINC-AN because the condition (10) is false. This result means that the LINC-AN condition and the LIMD condition are identical if we consider all possible strings in a population (practically, more than $O(2^k)$ strings).

From the above result, the LIMD condition has the same ability as the LINC-AN in identifying linkage with simpler conditions that require smaller number of comparisons.

6 Tightness Detection

Overlapping functions are considered difficult for linkage identification procedures because they may mislead them to obtain overspecified linkage groups. In this section, we propose an extension of the LIMD in order to identify correct linkage groups for overlapping functions. To detect overspecification of the obtained linkage sets, we introduce a *tightness* of linkage for each pair of loci. In the LIMD procedure, if `i` is in the `linkage_set[j]`, then `j` must be in the `linkage_set[i]`. However, this does not mean that `i` and `j` exist simultaneously in the other linkage sets `linkage_set[k]` ($k \neq i, j$). If the loci are tightly linked, they are expected to exist simultaneously in the other linkage sets. The tightness detection (TD) procedure we propose calculates tightness of each pair of loci by calculating the following:

$$\text{tightness}(i, j) = \frac{n1(i, j)}{n1(i, j) + n2(i, j)}, \tag{12}$$

where $n1(i, j)$ is the number of linkage sets that includes both i and j, and $n2(i, j)$ is the number of linkage sets that includes either i or j. The above equation calculates the ratio of simultaneous occurrence of the loci (i, j) in the obtained linkage sets. By definition, $0 \leq \text{tightness} \leq 1$.

To modify overspecified linkage sets, we remove loci `j` from `linkage_set[i]` when the following condition is satisfied:

$$\text{tightness}(i, j) < \delta, \tag{13}$$

where $0 \leq \delta \leq 1$. When $\delta = 1$, we allow only perfectly linked loci to be included in a linkage group.

The tightness detection (TD) procedure is shown in Appendix A. This procedure calculates $n1$ and $n2$ for each pair of loci (i, j) by scanning the linkage sets obtained by

the LIMD. We omit the initialization of n1, n2 (to be zero), because it is trivial. In the following, we denote the LIMD procedure with TD as the LIMD-TD that performs the TD procedure after the LIMD.

7 Population Sizing

The number of strings required to obtain correct linkage sets can be easily calculated in the same way as population sizing. In order-k delineable problems (Kargupta, 1995) that limit the problem difficulty at most the order of k, there exists at least one instance among 2^k order-k schemata that shows nonlinearity and non-monotonicity. Therefore, in the worst case, if we have only one string which shows nonlinearity/non-monotonicity, we need to check $O(2^k)$ strings for order-k delineable problems encoded into binary strings. More precisely, considering the worst case in which we have only one order-k schema which causes nonlinearity/non-monotonicity, the probability that we have a string with such schemata in a population of n strings is:

$$P = 1 - (1 - (1/2^k))^n \tag{14}$$

When we fix a success probability r, by solving $P = r$ we have:

$$n = \frac{\log(1-r)}{\log(1-1/2^k)} \simeq -2^k \log(1-r) \tag{15}$$

When we set $r = 1 - 2^{-k}$, at which a failure may occur in one of all the 2^k combinations of order-k schemata, we have:

$$n \simeq -2^k \log(1-r) = k2^k \tag{16}$$

On the other hand, in the best case, we need to check only one string to detect the linkage group. This happens when the entire string causes non-monotonicity inside the linkage. The number of locus pairs for a string length l is $\begin{pmatrix} l \\ 2 \end{pmatrix} \sim O(l^2)$. Therefore, the overall computational cost for the LINC and the LIMD procedures are the same, which need $O(l^2 2^k)$ function evaluations. Computational cost for the TD procedure is apparently $O(l^3)$, because the algorithm performs triple loops.

When we have noise in evaluating fitness values, we need to perform sampling to have more accurate estimation of fitness. In the following, we consider a fitness function with Gaussian noise defined as follows:

$$\tilde{f}(s) = f(s) + \delta, \text{ where } \delta \sim N(\mu, \sigma) \tag{17}$$

To modify the LIMD procedures to be robust to noise, we replace its fitness evaluation by the following averaging function.

$$f(s) \rightarrow \frac{1}{N} \sum_{j=1}^{N} \tilde{f}(s) \tag{18}$$

To estimate the effects on the LIMD conditions, we only consider the positive case in condition (5) (the negative case is essentially the same) and we define the amount of

violation of the condition as follows:

$$v_i(s) = \Delta f_i(s) - \Delta f_{ij}(s) \tag{19}$$
$$v_j(s) = \Delta f_j(s) - \Delta f_{ij}(s) \tag{20}$$

If $v_i > 0$ or $v_j > 0$, we detect a linkage. When the fitness function yields noise, the following two failures may occur.

Overspecification When we have a small negative value for v, a positive noise that exceeds v causes overspecified linkage because the LIMD detects a violation of the monotonicity condition to be considered as a linkage even though there is no violation actually.

Underspecification When we have a small positive value for v, a negative noise that exceeds v causes underspecified linkage.

When we have error $\delta_v \sim N(\mu_v, \sigma_v^2)$ for the violation of the conditions, the former case will occur when $\delta_v > v^-$, and the latter will occur when $-\delta_v > v^+$, where v^+ and v^- are the signal difference of v, the nearest values to the origin ($v = 0$) from positive and negative regions. To obtain a correct result, we need to satisfy $c\sigma_v < \min(v^-, v^+)$, where c is a scaling factor based on a given level of confidence.

Consider the case of noisy fitness functions. We have the following results for the amount of change in fitness for perturbations:

$$\Delta \tilde{f}_i = \Delta f_i + (\delta_1 - \delta_2) \tag{21}$$
$$\Delta \tilde{f}_j = \Delta f_j + (\delta_3 - \delta_4) \tag{22}$$
$$\Delta \tilde{f}_{ij} = \Delta f_{ij} + (\delta_5 - \delta_6), \tag{23}$$

where δ_i is independent Gaussian noise that follows $N(\mu, \sigma)$.

Therefore, we have the amount of violations for this noisy fitness as follows:

$$\tilde{v}_i(s) = v_i(s) + (\delta_1 - \delta_2) - (\delta_5 - \delta_6) \tag{24}$$
$$\tilde{v}_j(s) = v_j(s) + (\delta_3 - \delta_4) - (\delta_5 - \delta_6) \tag{25}$$

Consequently, we have the following results:

$$\tilde{v}_i(s) = v_i(s) + N(0, 4\sigma^2) \tag{26}$$
$$\tilde{v}_j(s) = v_j(s) + N(0, 4\sigma^2) \tag{27}$$

The mean μ of the noise distribution is canceled and there exists four times more variance than noise. The above results mean that we have an error that follows $N(0, 4\sigma^2)$ for the LIMD procedure. We can reduce this error by averaging fitness values. When we calculate the average of N fitness evaluations for each string, we can reduce the error to $N(0, 4\sigma^2/N)$. Therefore, we have $\mu_v = 0$, $\sigma_v = 2\sigma/\sqrt{N}$ and we need to satisfy the following condition to perform a reliable detection of the linkage set:

$$c\frac{2\sigma}{\sqrt{N}} < \min(v^+, v^-), \tag{28}$$

where c is the scaling factor. By solving Equation 28, we obtain the following result concerning the number of sampling:

$$N > \frac{4c^2\sigma^2}{(\min(v^+, v^-))^2} \tag{29}$$

From Equation 15, the total number of function evaluations necessary to obtain correct linkage groups is:

$$n_f > -2^k \log(1-r)\frac{4c^2\sigma^2}{(\min(v^+, v^-))^2} \tag{30}$$

The number of function evaluations is proportional to the variance of noise and inversely proportional to the square of the minimum difference from the border of the condition. Note that the above estimation is a rather conservative one because we only consider the worst case.

8 Empirical Results

We perform experiments on non-overlapping and overlapping test functions. For non-overlapping functions, we check the effectiveness of the LIMD for the sum of GA-difficult subfunctions and for a nonlinear function of the sum. We also show the equality of the LIMD and the LINC-AN empirically, and then check the validity of the population sizing for noisy fitness functions. For overlapping functions, we apply the LIMD and the LIMD-TD to the sum of GA-difficult subfunctions with parity overlapping factors. We show that the LIMD produces overspecified linkage groups and the LIMD-TD procedure becomes necessary to obtain correct results.

8.1 Non-overlapping Functions

For a non-overlapping test function, we employ the sum of 10 order-5 trap functions (string length $l = 50$) defined as follows:

$$f(s) = \sum_{i=1}^{10} f_i(u_i). \tag{31}$$

$$f_i(u_i) = \begin{cases} 4 - u_i & \text{if } 0 \le u_i \le 4 \\ 5 & \text{if } u_i = 5 \end{cases} \tag{32}$$

where u_i is the number of ones (unitation) in each 5-bit substring of s. Figure 1 shows an output of the LIMD procedure when we employ 100 strings. `LS [i]` is a set of linkage group for the `i`-th locus.

Since the test function has linkage among loci in 5-bit subfunctions, the result shows that we obtain correct linkage groups. This is because we have enough population size. We perform linkage identification by the LIMD and the LINC-AN with undersized populations and plot the result (the ratio of linkage sets correctly identified) in Figure 2.

Apparently, there is no difference between them except a small amount of noise caused by random initializations. From Equation 15, the number of strings needed for a 90%

```
LS [0] : 1 2 3 4          LS [17] : 18 16 15 19     LS [34] : 31 33 32 30
LS [1] : 0 2 3 4          LS [18] : 15 16 17 19     LS [35] : 36 37 38 39
LS [2] : 1 0 3 4          LS [19] : 18 16 17 15     LS [36] : 35 37 38 39
LS [3] : 1 2 0 4          LS [20] : 21 22 23 24     LS [37] : 35 36 38 39
LS [4] : 1 2 3 0          LS [21] : 20 22 23 24     LS [38] : 35 36 37 39
LS [5] : 6 9 8 7          LS [22] : 20 21 23 24     LS [39] : 35 36 37 38
LS [6] : 5 7 8 9          LS [23] : 20 21 22 24     LS [40] : 42 41 43 44
LS [7] : 6 9 8 5          LS [24] : 20 21 22 23     LS [41] : 42 40 43 44
LS [8] : 6 9 5 7          LS [25] : 27 26 28 29     LS [42] : 40 41 43 44
LS [9] : 6 5 7 8          LS [26] : 27 25 28 29     LS [43] : 42 41 40 44
LS [10] : 11 12 13 14     LS [27] : 25 26 28 29     LS [44] : 42 41 40 43
LS [11] : 10 12 13 14     LS [28] : 27 26 25 29     LS [45] : 46 47 48 49
LS [12] : 10 11 13 14     LS [29] : 27 26 25 28     LS [46] : 45 47 48 49
LS [13] : 10 12 11 14     LS [30] : 31 33 32 34     LS [47] : 46 45 49 48
LS [14] : 10 12 13 11     LS [31] : 30 32 33 34     LS [48] : 46 45 49 47
LS [15] : 18 16 17 19     LS [32] : 31 33 30 34     LS [49] : 46 45 47 48
LS [16] : 18 15 17 19     LS [33] : 31 30 32 34
```

Figure 1: Linkage sets obtained for the sum of non-overlapping 5-bit trap functions.

success probability of linkage identification is $n = -2^k \log(1-r) = -2^5 \log(1-0.9) \simeq 106.3$. As shown in Figure 2, both algorithms achieve more than 90% success with 30 strings, much less than the worst case estimation. The reason why results in Figure 2 are better is that the estimation in Equation 15 is a conservative one which assumes that only one of 2^k schemata shows nonlinearity/non-monotonicity.

We also perform experiments on some nonlinear functions of order-5 trap functions (string length $l = 50$) such as:

$$f(s) = [\sum_{i=1}^{10} f_i(u_i)]^2, \; f(s) = a[\sum_{i=1}^{10} f_i(u_i)]^2 + b[\sum_{i=1}^{10} f_i(u_i)] \tag{33}$$

$$f(s) = k_1(f_1(u_1) + f_2(u_2))^2 + \cdots + k_5(f_9(u_9) + f_{10}(u_{10}))^2, \tag{34}$$

where $f_i(u_i)$ is the same as in Equation 32. For these functions, the LINC-AN and the LIMD generate essentially the same results as in Figure 1. When we employ the LINC without allowable nonlinearity, overspecified linkage groups are obtained due to the nonlinearity of the function. By employing the LIMD procedure, which considers GA-easy nonlinearity to be excluded from the linkage sets, we can obtain correct linkage sets for these nonlinear functions.

Although the estimation in Equation 29 is obtained almost directly from statistical theory, it is still important to verify its validity empirically. In this experiment, we employ the sum of trap functions with Gaussian noise defined as follows:

$$\tilde{f}(s) = \sum_{i=1}^{10} f_i(u_i) + N(0, \sigma^2) \tag{35}$$

$f_i(s)$ is the same as in Equation 32. We employ a population of 32 strings. We observe the ratio of linkage groups correctly identified by changing the amount of noise (by changing σ) and the number of sampling fitness values. Figure 3 shows the results of the

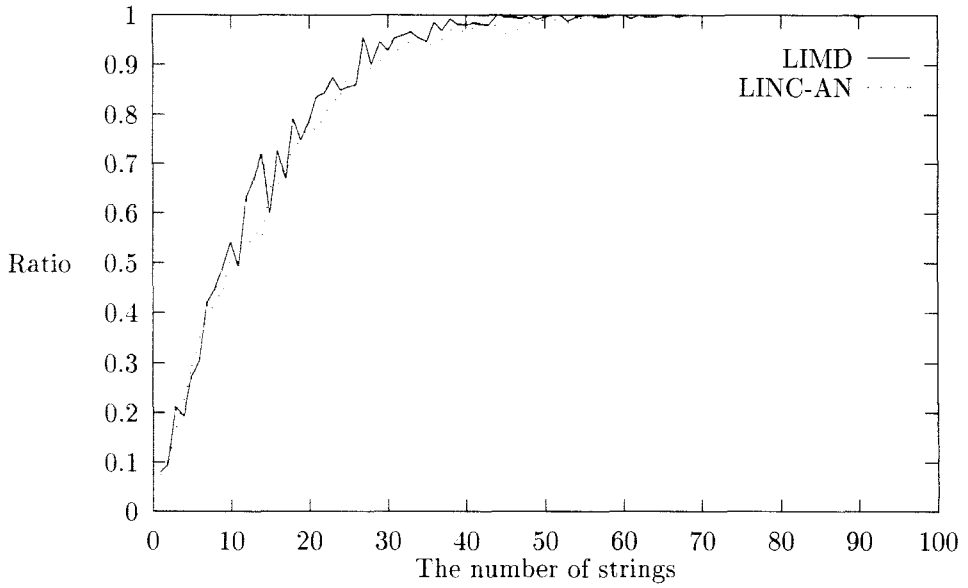

Figure 2: Ratio of correct linkage groups identified by the LINC-AN and the LIMD.

Table 1: Number of samples calculated from Equation 29.

σ of noise	# of samples (N)
0.25	1
0.5	4
1.0	16
1.5	36
2.0	64

experiment. The x-axis is the number N of samples in each string and the y-axis is the ratio of correct linkage sets. We plot the result for $\sigma = 0.25, 0.5, 1.0, 1.5, 2.0$.

In the function we employed, $v^+ = v^- = 1.0$ because the minimum difference between a pair of function values is 1.0. When we set the value $c = 2.0$ to achieve around 97.5% success rate, we obtain the lower bound of N from Equation 29 as shown in Table 1.

We can easily see the validity of the obtained values of N by comparing them with the result in Figure 3.

8.2 Overlapping Functions

We perform experiments on overlapping functions consisting of 5-bit trap functions loosely connected by parity functions. The purpose of the TD procedure is to remove such loose connections and obtain only tightly linked linkage groups inside the 5bit trap functions.

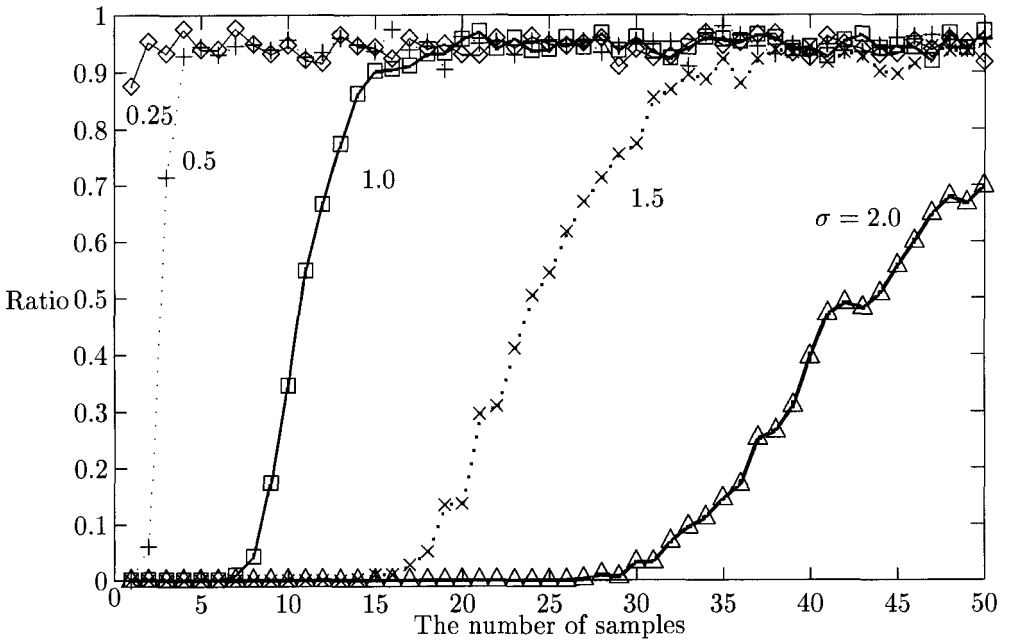

Figure 3: Ratio of correct linkage groups identified for noisy functions.

For the experiments, we employ a 5-bit overlapping function as follows:

$$f(s) = \sum_{i=1}^{10} [f_i(u_i) + \omega\phi(u_i + u_{i\ominus 1})], \tag{36}$$

and a 1-bit overlapping function,

$$f(s) = \sum_{i=1}^{10} [f_i(u_i) + \omega\phi(x_{5i} + x_{5i\ominus 1})], \tag{37}$$

where $\phi(x)$ is a parity function that outputs $+1$ when x is odd; -1 when x is even; and $\ominus$ is a minus operator of modular 10 in the 5-bit function (the number of subfunctions), or 50 for the 1-bit function (string length).

In the above functions, adjacent BBs or adjacent loci are connected by the parity function. The value ω represents the strength of this connection. When ω is small, the LIMD without TD can identify correct linkage groups because the parity does not affect the result of the non-monotonicity conditions. When ω is large enough, however, we cannot ignore the effect of the overlapping parity function. For example, we show the results when $\omega = 2.0$ in Figure 4 for the 5-bit overlapping function and in Figure 5 for the 1-bit overlapping function. In the experiments, we employ an appropriately sized population with 100 strings.

```
LS [0] : 5 6 7 8 9 45 46 47 48 49 1 2 3 4
LS [1] : 5 6 7 8 9 45 46 47 48 49 0 2 3 4
LS [2] : 5 6 7 8 9 45 46 47 48 49 1 0 3 4
LS [3] : 5 6 7 8 9 45 46 47 48 49 1 0 2 4
LS [4] : 5 6 7 8 9 45 46 47 48 49 1 0 2 3
LS [5] : 0 1 2 3 4 10 11 12 13 14 6 7 8 9
LS [6] : 0 1 2 3 4 10 11 12 13 14 5 7 8 9
LS [7] : 0 1 2 3 4 10 11 12 13 14 5 6 8 9
LS [8] : 0 1 2 3 4 10 11 12 13 14 5 6 7 9
LS [9] : 0 1 2 3 4 10 11 12 13 14 5 6 7 8
LS [10] : 5 6 7 8 9 11 12 13 14 15 16 17 18 19
LS [11] : 5 6 7 8 9 10 12 13 14 15 16 17 18 19
LS [12] : 5 6 7 8 9 10 11 13 14 15 16 17 18 19
LS [13] : 5 6 7 8 9 10 11 12 14 15 16 17 18 19
LS [14] : 5 6 7 8 9 10 11 12 13 15 16 17 18 19
LS [15] : 10 11 12 13 14 20 21 22 23 24 16 17 18 19
LS [16] : 10 11 12 13 14 20 21 22 23 24 15 17 18 19
LS [17] : 10 11 12 13 14 20 21 22 23 24 15 16 18 19
LS [18] : 10 11 12 13 14 20 21 22 23 24 15 16 17 19
LS [19] : 10 11 12 13 14 20 21 22 23 24 15 16 17 18
LS [20] : 15 16 17 18 19 21 22 23 24 25 26 27 28 29
LS [21] : 15 16 17 18 19 20 25 26 27 28 29 22 23 24
LS [22] : 15 16 17 18 19 20 25 26 27 28 29 21 23 24
LS [23] : 15 16 17 18 19 20 25 26 27 28 29 21 22 24
LS [24] : 15 16 17 18 19 20 25 26 27 28 29 21 22 23
LS [25] : 20 21 22 23 24 26 27 28 29 30 31 32 33 34
LS [26] : 20 21 22 23 24 25 27 28 29 30 31 32 33 34
LS [27] : 20 21 22 23 24 25 26 28 29 30 31 32 33 34
LS [28] : 20 21 22 23 24 25 26 27 29 30 31 32 33 34
LS [29] : 20 21 22 23 24 25 26 27 28 30 31 32 33 34
LS [30] : 25 26 27 28 29 31 32 33 34 35 36 37 38 39
LS [31] : 25 26 27 28 29 30 32 33 34 35 36 37 38 39
LS [32] : 25 26 27 28 29 30 31 33 34 35 36 37 38 39
LS [33] : 25 26 27 28 29 30 31 32 34 35 36 37 38 39
LS [34] : 25 26 27 28 29 30 31 32 33 35 36 37 38 39
LS [35] : 30 31 32 33 34 36 37 38 39 40 41 42 43 44
LS [36] : 30 31 32 33 34 35 37 38 39 40 41 42 43 44
LS [37] : 30 31 32 33 34 35 36 38 39 40 41 42 43 44
LS [38] : 30 31 32 33 34 35 36 37 39 40 41 42 43 44
LS [39] : 30 31 32 33 34 35 36 37 38 40 41 42 43 44
LS [40] : 35 36 37 38 39 41 42 43 44 45 46 47 48 49
LS [41] : 35 36 37 38 39 40 42 43 44 45 46 47 48 49
LS [42] : 35 36 37 38 39 40 41 43 44 45 46 47 48 49
LS [43] : 35 36 37 38 39 40 41 42 44 45 46 47 48 49
LS [44] : 35 36 37 38 39 40 41 42 43 45 46 47 48 49
LS [45] : 0 1 2 3 4 40 41 42 43 44 46 47 48 49
LS [46] : 0 1 2 3 4 40 41 42 43 44 45 47 48 49
LS [47] : 0 1 2 3 4 40 41 42 43 44 45 46 48 49
LS [48] : 0 1 2 3 4 40 41 42 43 44 45 46 47 49
LS [49] : 0 1 2 3 4 40 41 42 43 44 45 46 47 48
```

Figure 4: Linkage sets obtained for the 5-bit overlapping function ($\omega = 2.0$) by the LIMD.

```
LS [0] : 49 1 4 2 3
LS [1] : 0 2 3 4
LS [2] : 1 0 3 4
LS [3] : 1 2 0 4
LS [4] : 5 1 0 2 3
LS [5] : 4 6 7 8 9
LS [6] : 5 7 8 9
LS [7] : 6 5 8 9
LS [8] : 6 5 7 9
LS [9] : 10 6 5 7 8
LS [10] : 9 14 13 11 12
LS [11] : 10 12 14 13
LS [12] : 10 11 13 14
LS [13] : 10 12 14 11
LS [14] : 15 10 11 13 12
LS [15] : 14 18 16 17 19
LS [16] : 18 19 17 15
LS [17] : 18 16 19 15
LS [18] : 15 16 17 19
LS [19] : 20 18 16 17 15
LS [20] : 19 21 22 23 24
LS [21] : 20 24 22 23
LS [22] : 20 23 21 24
LS [23] : 20 24 22 21
LS [24] : 20 25 23 21 22
LS [25] : 24 27 29 26 28
LS [26] : 27 25 28 29
LS [27] : 26 25 28 29
LS [28] : 27 25 26 29
LS [29] : 30 27 25 26 28
LS [30] : 29 32 31 33 34
LS [31] : 30 32 33 34
LS [32] : 30 31 33 34
LS [33] : 31 34 30 32
LS [34] : 35 31 33 32 30
LS [35] : 34 36 37 38 39
LS [36] : 37 35 38 39
LS [37] : 36 35 38 39
LS [38] : 35 36 37 39
LS [39] : 40 35 36 37 38
LS [40] : 39 44 42 41 43
LS [41] : 42 44 40 43
LS [42] : 41 43 40 44
LS [43] : 42 44 41 40
LS [44] : 45 40 41 42 43
LS [45] : 44 46 47 48 49
LS [46] : 45 47 48 49
LS [47] : 45 46 48 49
LS [48] : 49 45 46 47
LS [49] : 0 48 45 46 47
```

Figure 5: Linkage sets obtained for the 1-bit overlapping function ($\omega = 2.0$) by the LIMD.

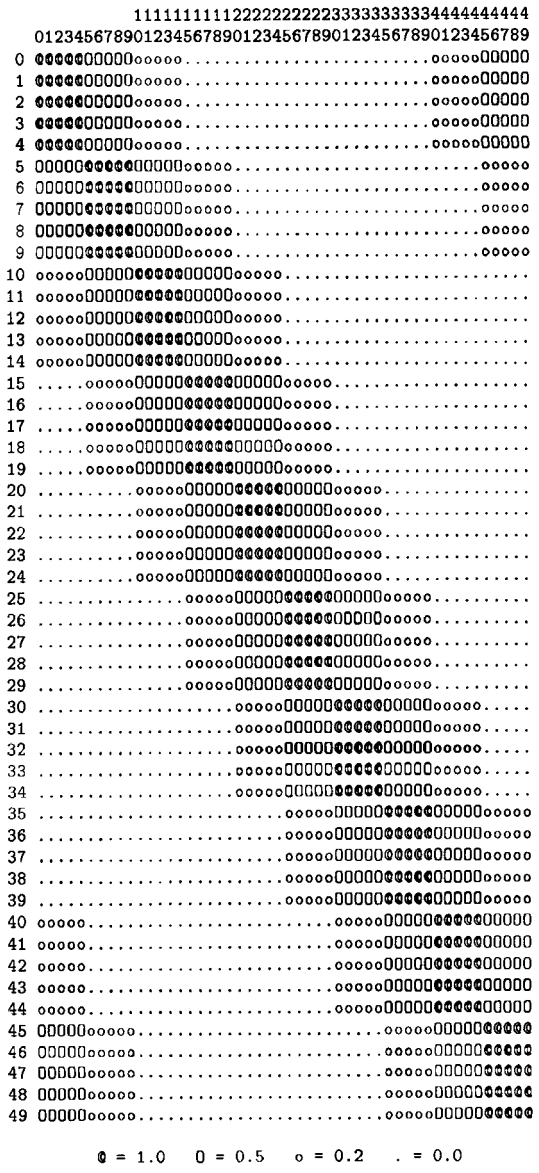

Figure 6: Tightness between loci for the 5-bit overlapping function ($\omega = 2.0$).

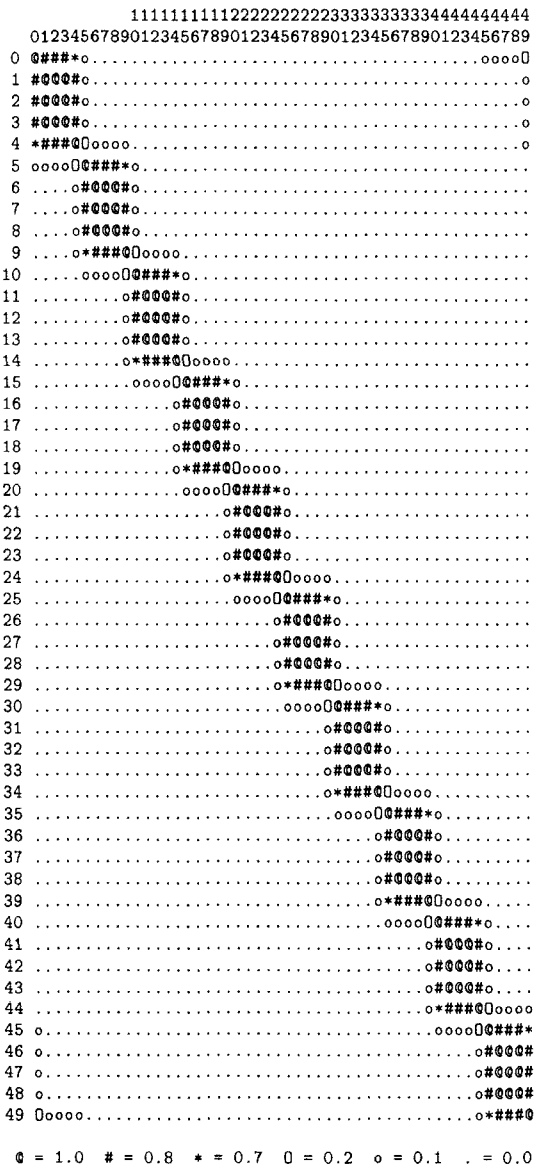

Figure 7: Tightness between loci for the 1-bit overlapping function ($\omega = 2.0$).

```
LS [0] : 1 2 3 4
LS [1] : 0 2 3 4
LS [2] : 1 0 3 4
LS [3] : 1 0 2 4
LS [4] : 1 2 3 0
LS [5] : 6 7 8 9
LS [6] : 5 7 8 9
LS [7] : 5 6 8 9
LS [8] : 5 6 7 9
LS [9] : 5 6 7 8
LS [10] : 11 12 13 14
LS [11] : 10 12 13 14
LS [12] : 10 11 13 14
LS [13] : 10 11 12 14
LS [14] : 10 11 12 13
LS [15] : 16 17 18 19
LS [16] : 15 17 18 19
LS [17] : 15 16 18 19
LS [18] : 15 16 17 19
LS [19] : 15 16 17 18
LS [20] : 21 22 23 24
LS [21] : 20 22 23 24
LS [22] : 20 21 23 24
LS [23] : 20 21 22 24
LS [24] : 20 21 22 23
LS [25] : 26 27 28 29
LS [26] : 25 28 29 27
LS [27] : 25 28 29 26
LS [28] : 25 26 27 29
LS [29] : 25 26 27 28
LS [30] : 31 32 33 34
LS [31] : 30 32 33 34
LS [32] : 30 31 33 34
LS [33] : 30 31 32 34
LS [34] : 30 31 32 33
LS [35] : 36 37 38 39
LS [36] : 35 38 39 37
LS [37] : 35 38 39 36
LS [38] : 35 36 37 39
LS [39] : 35 36 37 38
LS [40] : 41 42 43 44
LS [41] : 40 42 43 44
LS [42] : 40 41 43 44
LS [43] : 40 41 42 44
LS [44] : 40 41 42 43
LS [45] : 46 47 48 49
LS [46] : 45 47 48 49
LS [47] : 45 46 48 49
LS [48] : 45 46 47 49
LS [49] : 45 46 47 48
```

Figure 8: Linkage sets obtained for the 5-bit overlapping function ($\omega = 2.0$) by the LIMD-TD.

```
LS [0] : 1 3 2 4
LS [1] : 2 3 4 0
LS [2] : 1 3 4 0
LS [3] : 1 2 0 4
LS [4] : 1 2 3 0
LS [5] : 6 7 8 9
LS [6] : 7 8 5 9
LS [7] : 6 5 8 9
LS [8] : 6 5 7 9
LS [9] : 5 6 7 8
LS [10] : 14 13 11 12
LS [11] : 10 12 14 13
LS [12] : 10 11 13 14
LS [13] : 10 12 11 14
LS [14] : 10 11 13 12
LS [15] : 16 17 18 19
LS [16] : 18 19 17 15
LS [17] : 18 16 15 19
LS [18] : 16 17 19 15
LS [19] : 18 16 15 17
LS [20] : 21 22 23 24
LS [21] : 20 22 23 24
LS [22] : 20 23 21 24
LS [23] : 20 24 22 21
LS [24] : 20 23 21 22
LS [25] : 27 26 28 29
LS [26] : 27 25 28 29
LS [27] : 26 28 25 29
LS [28] : 27 25 26 29
LS [29] : 27 25 26 28
LS [30] : 32 33 34 31
LS [31] : 32 33 34 30
LS [32] : 30 31 33 34
LS [33] : 31 30 32 34
LS [34] : 31 33 30 32
LS [35] : 36 37 38 39
LS [36] : 37 35 38 39
LS [37] : 36 35 38 39
LS [38] : 35 36 37 39
LS [39] : 37 35 36 38
LS [40] : 42 41 44 43
LS [41] : 42 44 40 43
LS [42] : 41 43 40 44
LS [43] : 42 44 41 40
LS [44] : 41 42 43 40
LS [45] : 46 47 48 49
LS [46] : 45 47 48 49
LS [47] : 45 46 48 49
LS [48] : 49 45 46 47
LS [49] : 48 45 46 47
```

Figure 9: Linkage sets obtained for the 1-bit overlapping function ($\omega = 2.0$) by the LIMD-TD.

Apparently, these results indicate overspecified linkage groups were obtained. For the test functions, we obtained correct results (as in Figure 1) when $\omega < 0.5$ and, otherwise, overspecified ones as in the above. Since the range of the parity function is $[-1, 1]$, the overall effect to the function becomes $2 \times \omega$. In the non-overlapping test functions, the minimum difference of the fitness function is 1.0, therefore we have $\omega < 0.5$ by solving $2\omega < 1.0$.

By employing the TD procedure, we can detect tightness between each pair of loci in the linkage sets obtained by the LIMD. From results in Figures 4 and 5, we can see that some pairs of loci exist in the same linkage set and others do not. For example, locus 1 and locus 2 always appear in the same linkage sets but locus 1 and locus 10 do not. The basic idea of the TD procedure is to detect the "tightness" of simultaneous existence in order to find tight linkages. Figures 6 and 7 show tightness matrices for the 5-bit and the 1-bit overlapping functions calculated from the obtained linkage sets. In the figures, a number assigned in each row or column represents an ID of a locus (from 0 to 49) and a matrix of characters consisting of { `@`, `*`, `#`, `0`, `o`, `.` } represents a matrix of tightness values for pairs of loci.

From the tightness matrices in the figures, we can easily identify the effect of parity overlapping functions that loosely connects a group of loci which do not belong to a BB. By removing such loci from the linkage sets, we obtain accurate linkage sets. The results in Figure 8 and Figure 9 show the linkage sets obtained by applying the TD procedure for the 5-bit and the 1-bit overlapping functions. We applied the LIMD and then the TD procedures to a population of 100 binary strings randomly initialized. For the overlapping test functions, we employ $\delta = 0.6$ as a threshold in Equation 13.

The results indicate that correct linkage groups are identified by removing unnecessary loci based on their tightness. The LIMD-TD procedure is robust to overlapping effects; that is, even when the value of ω becomes larger, it generates the same result. This is because the TD condition considers a meta-level relation among loci in the linkage sets obtained from the monotonicity conditions and does not deal with the change of fitness values directly.

9 Conclusion

In this paper, we have discussed direct linkage identification procedures based on detections of nonlinearity or non-monotonicity. To obtain more accurate linkage groups, we also proposed a tightness detection procedure that removes overspecified, loosely connected linkage groups. Through experiments on linear/nonlinear and non-overlapping/overlapping test functions, we showed that the LIMD can identify correct linkage sets for non-overlapping functions and their nonlinear functions. The LIMD-TD can identify more accurate linkage groups even for overlapping functions which are considered difficult for the linkage identification procedures. Although the proposed procedures are not considered perfect in detecting the "true" linkage groups of a problem, the obtained linkage groups indicate that the problem can be decomposed by linkage groups into small subproblems and there is no reason for us to ignore such important information. Concerning computational complexity, the LIMD needs to check a violation of the monotonicity condition for $O(2^k)$ strings to obtain accurate results with a fixed success probability. Since the computational cost to check all the pair of loci is $O(l^2)$, overall complexity of the LIMD is $O(l^2 2^k)$. The cost for the TD is $O(l^3)$ which is not dependent upon the population size.

Appendix A: Procedures

```
Procedure: Linkage Identification by non-Monotonicity Detection (LIMD)


for(all s in a population) {
  for(i = 0; i < length; i++) {
    s' = Perturb(s, i);
    df1 = f(s') - f(s);
    for(j = i; j < length; j++) {
      if(i != j) {
        s' = Perturb(s, j);
        df2 = f(s') - f(s);
        s'' = Perturb(s', i)
        df12 = f(s'') - f(s);
        if(df1 > 0 & df2 > 0) {
          if(df12 > df1 & df12 > df2) {
            /* do nothing */
          }
          else {
            adding j to linkage_set[i];
            adding i to linkage_set[j];
          }
        }
        if(df1 < 0 & df2 < 0) {
          if(df12 < df1 & df12 < df2) {
            /* do nothing */
          }
          else {
            adding j to linkage_set[i];
            adding i to linkage_set[j];
          }
        }
      }
    }
  }
}
```

```
Procedure: Tightness Detection (TD)


for(i = 0; i < length; i++) {
  adding i to linkage_set[i];
  for(j = i; j < length; j++) {
    if(i != j) {
      for(k = 0; k < length; k++) {
        if(i and j exist in linkage_set[k]) {
          n1[i][j]++;
        }
        else if(neither i nor j exist in linkage_set[k] {
          /* do nothing */
        }
        else {
          n2[i][j]++;
        }
      }
      if(n1[i][j] != 0 or n2[i][j] != 0) {
        tightness[i][j] = n1[i][j]/(n1[i][j] + n2[i][j]);
      }
      else {
        tightness[i][j] = 0.0;
      }
    }
    else { /* if i == j */
      tightness[i][j] = 1.0;
    }
  }
}

for(i = 0; i < length; i++) {
  for(each entry j in linkage_set[i]) {
    if(tightness[i][j] < delta) {
      remove j from linkage_set[i];
    }
  }
}
```

Acknowledgments

Dr. Munetomo initiated this work as a visiting scholar at the Illinois Genetic Algorithms Laboratory.

Professor Goldberg's contribution to this paper was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-97-1-0050. Research funding for this project was also provided by a grant from the U.S. Army Research Laboratory under the Federal Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S Army, the Air Force Office of Scientific Research or the U. S. Government.

References

Bandyopadhyay, S., Kargupta, H. and Wang, G. (1998). Revisiting the GEMGA: Scalable evolutionary optimization through linkage learning. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 603–608, IEEE Service Center, Piscataway, New Jersey.

Corno, F., Sonza Reorda, M. and Squillero, G. (1998). A new evolutionary algorithm inspired by the selfish gene theory. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 575–580, IEEE Service Center, Piscataway, New Jersey.

Goldberg, D. E., Deb, K. and Thierens, D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instrument and Control Engineers*, 32(1):10–16.

Harik, G. R. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. Unpublished doctoral dissertation, University of Michigan, Ann Arbor, Michigan. Also IlliGAL Report 97005.

Kargupta, H. (1995). SEARCH, polynomial complexity, and the fast messy genetic algorithm. Technical Report 95008, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Kargupta, H. (1996a). The gene expression messy genetic algorithm. In *Proceedings of 1996 IEEE International Conference on Evolutionary Computation*, pages 814–819, IEEE Service Center, Piscataway, New Jersey.

Kargupta, H. (1996b). The performance of the gene expression messy genetic algorithm on real test functions. In *Proceedings of 1996 IEEE International Conference on Evolutionary Computation*, pages 631–636, IEEE Service Center, Piscataway, New Jersey.

Kargupta, H. (1996c). SEARCH, evolution, and the gene expression messy genetic algorithm. Unclassified Report LA-UR 96-60, Los Alamos National Laboratory, Los Alamos, New Mexico.

Kargupta, H., Goldberg, D. E. and Wang, L. (1997). Extending the class of order-k delineable problems for the gene expression messy genetic algorithm. *Genetic Programming 1997, Proceedings of the Second Annual Conference*, pages 364–369, Morgan Kaufmann, San Francisco, California.

Kazadi, S. T. (1997). Conjugate schema in genetic search. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 10–17, Morgan Kaufmann, San Francisco, California.

Mühlenbein, H. (1997). The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346.

Mühlenbein, H. and Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature, PPSN IV*, pages 178–187, Springer-Verlag, Berlin, Germany.

Munetomo, M. and Goldberg, D. E. (1998). Identifying linkage by nonlinearity check. IlliGAL Report 98012, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Pelikan, M., Cantú-Paz, E. and Goldberg, D. E. (1998). Linkage problem, distribution estimation, and Bayesian networks. IlliGAL Report 98013, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Pelikan, M. and Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. In Roy, R., Furuhashi, T. and Chawdhry, P. K., editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535, Springer-Verlag, London, England.

Thierens, D. (1995). *Analysis and design of genetic algorithms*. Doctoral dissertation, Katholieke Universiteit Leuven, Leuven, Belgium.

Thierens, D. and Goldberg, D. E. (1993). Mixing in genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 38–45, Morgan Kaufmann, San Francisco, California.

Winter, P. C., Hickey, G. I. and Fletcher, H. L. (1998). *Instant Notes in Genetics*. Springer-Verlag, New York, New York.