Population Sizing of Dependency Detection by Fitness Difference Classification

Miwako Tsuji, Masaharu Munetomo, and Kiyoshi Akama

Hokkaido University, North 11 West 5, Sapporo, 060-0811, Japan
m_tsuji@cims.hokudai.ac.jp, {munetomo,akama}@iic.hokudai.ac.jp

Abstract. Recently, the linkage problem has attracted attention from researchers and users of genetic algorithms and many efforts have been undertaken to learn linkage. Especially, (1) perturbation methods (PMs) and (2) estimation of distribution algorithms (EDAs) are well known and frequently employed for linkage identification. In our previous work [TMA04], we have proposed a novel approach called Dependency Detection for Distribution Derived from *df* (D^5) which inherits characteristics from both EDAs and PMs. It detects dependencies of loci by estimating the distributions of strings classified according to fitness differences and can solve EDA difficult problems requiring a smaller number of fitness evaluations. In this paper, we estimate population size for the D^5 and its computation cost. The computation cost slightly exceeds $O(l)$, which is less than the PMs and some of EDAs.

1 Introduction

A set of loci tightly linked to form a building block is called a linkage set and encoding such loci loosely results building block disruptions. Several efforts have been undertaken to ensure appropriate building block processing without prior knowledge of problem. Two major methods of them are follows:

1. Perturbation Methods (PMs)
2. Estimation of Distribution Algorithms (EDAs)

PMs examine fitness differences by perturbations at loci to detect interdependency among them. They can recognize building blocks with lower marginal fitness contributions, but require a large number of extra fitness evaluations in addition to the usual fitness evaluations which are performed to select strings. For example, the LINC [MG99] requires $O(l^2)$ fitness evaluations for its linkage identification where l is string length. Heckendorn et al.[HW03] shows algorithm which uses the Walsh coefficients. This algorithm behaves similar to the LINC when it considers order-2 dependencies. But while the LINC guesses order-3 or more dependencies, it introduces higher-order perturbations (probes). EDAs like the BOA [PGCP99] employ probabilistic modeling of promising solutions to generate new solutions instead of the crossover and mutation operators of simple GAs. Some of EDAs are based on conditional probabilities to model dependency

A.H. Wright et al. (Eds.): FOGA 2005, LNCS 3469, pp. 282–299, 2005.

of variables. They can construct their models without additional fitness evaluations, however, it is difficult for EDAs to recognize low scaling building blocks. For such problems, EDAs need more strings and generations, therefore, the total number of evaluations increases to that of the PMs.

If a problem is composed of variously scaled sub-problems, fitness of the problem is dominated by solutions of highly scaled sub-problems (i.e. important building blocks) and GAs should focus only on the sub-problems. The scale of sub-problems means the amount of contribution of each sub-problems. The highly scaled sub-problems gives large contribution to fitness, while lowly scaled sub-problems gives small contribution to fitness. Therefore, GAs solve sub-problems sequentially from those scaled larger to those scaled smaller, and sometimes, the low scaled building blocks are lost while they are on a waiting list. For example, consider maximizing a problem of 4 variables, $f(s_1, s_2, s_3, s_4) = g(s_1, s_3) + h(s_2, s_4)$ where $s_1, \cdots s_4$ are variables. If $g_{\max}(s_1, s_3) = 5$ and $h_{\max}(s_2, s_4) = 5$, $h(s_3, s_2)$ and $g(s_2, s_4)$ are searched in parallel. On the other hand, if $g_{\max}(s_1, s_3) = 7$ and $h_{\max}(s_2, s_4) = 3$, GAs focus on larger scaled sub-problem, $g(s_1, s_3)$ first and then give their eyes to $h(s_2, s_4)$. This sequential search procedure is referred as domino convergence [TGP98,LGP00].

In our previous work [TMA04], we have proposed another approach called the Dependency Detection for Distribution Derived from *df* (D^5) which combines both features of the previous methods. It detects dependencies of loci by estimating the distributions of strings classified according to fitness differences. Generally, EDAs estimate bias in selected sub-population and such bias come naturally from selection according to fitness. On the other hand, the D^5 makes sub-population biased artificially by perturbations. Therefore, the D^5 can solve EDA difficult problems using less computation cost than the PMs. The experiments showed that it reduces computation cost considerably and the number of evaluations is approximately $O(l)$ where l is string length.

In this paper, we estimate the growth of the required population size for the D^5 theoretically in order to understand scalability of the D^5. The number of strings is an important factor for linkage identification quality and computation cost. Resulting number of evaluations exceeds $O(l)$ slightly but still far less than PMs and even some of EDAs.

This paper is organized as follows. First, we show a brief introduction to the D^5. And its population sizing is discussed in section 4. After theoretical estimation of population size, some numerical experiments are performed to confirm the result in section 5 and finally this paper is concluded in section 7.

2 Background

Decomposability is one of the grounds for the advantage of genetic algorithm over random search. Additively decomposable functions are one of representations of the decomposable problem. An additively decomposable function is defined as follows:

$$f(s) = \sum_{\forall v \in V} f_v(s) \tag{1}$$

where s is a string, v is a set of loci that composing a sub-function (i.e a linkage set), $f_v(s)$ is a sub-function defined over v, and V is a set of disjoint sets of loci (i.e. a set of linkage sets). We consider only bit string as s. We consider only bit string as s. In addition, we assume that the sub-functions do not overlap each other, i.e. $v \cap v' = \emptyset$ $(v, v' \in V)$. The length of s is denoted by l and sum of the number of loci in each v is equal to l:

$$\sum_{v \in V} |v| = l \tag{2}$$

We limit maximum size of a linkage set v to k.

$$|v| \leq k \quad \forall v \in V \tag{3}$$

k is known as order of (sub-)problem or order of building block. The additively decomposable functions are known as order k delineable problems defined by Kargupta [Kar95]. In this paper, we consider the linkage set is the set of loci which are linked and are not separable. If a building blocks can be constructed by crossover, these loci should be separable that can be optimized separately.

3 Dependency Detection by Fitness Differences

In this section, we show a brief explanation of the Dependency Detection for Distribution Derived from *df* (D^5). The D^5 combines PMs and EDAs in order to obtain bias of sub-population to be estimated rapidly even if there are some low scaling sub-problems. It detects dependencies of loci by estimating the distributions of strings classified according to fitness differences.

EDAs learn problem structure from bias of sub-population and such bias is given by selection pressure based on fitness. Therefore, if problem is composed of variously scaled sub-problems, then modeling processes in EDAs focus only on highly-scaled sub-problems. On the other hand, the D^5 makes sub-population to be estimated biased artificially by perturbations. Therefore, the D^5 can detect such EDA-difficult sub-problems. Moreover, although the D^5 requires additional fitness evaluations due to calculate fitness differences, the number of evaluation is less than the PMs which generally perform pairwise perturbations because the D^5 employs estimation instead of higher order perturbations.

3.1 Algorithm

Genetic algorithm using the D^5 is composed of two parts (1) detecting linkage sets and (2) generating, increasing and combining building blocks. In this paper, we concentrate on the first part because if we know problem structures we can perform subsequent optimization processes easily and efficiently.

Fig. 1 shows the algorithm of the D^5. The algorithm consists of three parts: (1) calculating of fitness differences, (2) classifying of strings according to the differences and (3) estimating of the classified strings. After initializing population, following procedures are repeated for each locus i: At first, locus i in

1. initialize population with n strings
2. for each locus i
   (a) calculate fitness difference $df_i(s^p)$ by a perturbation at locus i in string s^p $(p = 1, 2, \cdots, n)$.
   (b) classify strings according to their fitness differences into sub-populations.
   (c) estimate sub-populations and construct linkage sets (see Fig. 2 for detail).

Fig. 1. Overall Algorithm of linkage identification in the D^5

1. for each sub-population p classified by the Classification Algorithm
   (a) initialize set of loci $v_1 = \{1, 2, \cdots, i-1, i+1, \cdots, l\}$ and $v_2 = \{i\}$
   (b) while $|v_2| < K$, where K is pre-defined problem complexity
      i. calculate a entropy $E_j = E(v_2 \cup \{j\})$ for all locus $j \in v_1$
      ii. $h = \arg\min_{j \in v_1} E_j$
      iii. update $v_1 = v_1 - \{h\}$ and $v_2 = v_2 \cup \{h\}$
   (c) $v_p = v_2$ and $E_p = E(v_2)$
2. select v_p with the smallest E_p as the linkage set for locus i

Fig. 2. Construct Linkage Set

each string s^p is perturbed and then fitness difference for the perturbation is calculated as follows:

$$df_i(s^p) = f(s^p) - f(s_i^p) \tag{4}$$

In the above equation, s_i^p is a string perturbed at locus i. Then, strings are classified into sub-populations according to their fitness differences $df_i(s^p)$. We employed a simple centroid method for classification, but other approaches like k-means can also be applied. In this method, the centroid of a cluster is determined by averaging $df_i(s^p)$ of all strings within that cluster. The distance between two clusters is defined as the distance between the centroids of the clusters. The pair of clusters with the smallest distance is merged until a termination criteria is satisfied. If the smallest distance of all the rest exceeds a threshold, the classification is terminated. The threshold should be small for problems which consist of independent sub-problems, while it should be large for those with interacted sub-problems.

Finally, the sub-populations are estimated in order to detect loci which depend on locus i.

Fig. 2 is the algorithm to construct a linkage set for locus i. First, a set v_2 is initialized as $\{i\}$. The locus j which gives the smallest entropy $E(v_2 \cup \{j\})$ is merged repeatedly until the size of linkage set exceeds problem complexity k. This defines the order of a sub-problem and is given by algorithm users. The order of a building block is equal to k because we assume the building block is the optimal solution of the sub-problem. The entropy measure is defined as

$$E(v_2) = -\sum_{x=1}^{2^{|v_2|}} p_x \log_2 p_x, \tag{5}$$

where p_x is the appearance ratio of each schema x and $2^{|v_2|}$ is the number of all possible schema defined by v_2. This procedure is applied to all sub-populations except those including small number of strings. The sub-populations having small number of strings are ignored because estimating distribution of small samples has risk of unreliable result. The population sizing in section 4 will give the threshold for the sampling size. After the estimation of all sub-populations, the linkage set v_2 which gives the smallest entropy $E(v_2)$ is selected as linkage set v_i of locus i.

3.2 Example

As example, we use sum of the order 3 deceptive problem which was used as an opponent of the messy GA[GKD89] and defined as follows:

$$\begin{aligned} f(s) &= \sum_{\forall v \in V} f_v(s) \\ f_v(s) &= 30 \text{ if } 111, \quad 0 \text{ if } 110, 101, 011, \\ &\quad\ 14 \text{ if } 100, \quad 22 \text{ if } 010, \quad 26 \text{ if } 001, \quad 28 \text{ if } 000 \end{aligned} \tag{6}$$

where $V = \{\{1,2,3\},\{4,5,6\}\}$ and $f_v(s)$ is defined by each schema of $\{1,2,3\}$ and $\{4,5,6\}$. Fig. 3 shows the perturbation in the 1st locus. In this figure, strings having $df_1 = 30$ are belong to a sub-population. In the sub-population, linkage set $\{1,2,3\}$ has only schema 011 and $E(\{1,2,3\})$ should be zero. On the other hand, linkage set $\{1,4,5\}$ has schemata $010, 001, 010, 011$ and $E(\{1,4,5\})$ should be relatively large. Therefore the algorithm evaluates that a relationship between locus 1, 2, and 3 take place more likely than a relationship between locus 1, 4, and 5.

s	$f(s)$	s_1	$f(s_1)$	bias	$df_1(s)$
011100	14	111100	44	011***	30
011010	27	111010	57	011***	30
011101	0	111101	30	011***	30
011110	0	111110	30	011***	30
011110	0	111110	30	011***	30
		...			

s	$f(s)$	s_1	$f(s_1)$	bias	$df_1(s)$
110101	0	010101	27	110***	27
110011	0	010011	27	110***	27
110001	26	010001	53	110***	27
101100	14	001100	40	101***	26
101101	0	001101	26	101***	26
		...			

Fig. 3. Strings classified according to df_1

If problems are (quasi-)decomposable like $f(s) = \sum_{v \in V} f_v(s)$ then fitness differences for perturbation in locus i are calculated as

$$df_i(s) = f(s) - f(s_i)$$

$$= [f_{\hat{v}}(s) + \sum_{v \neq \hat{v}, v \in V} f_v(s)] - [f_{\hat{v}}(s_i) + \sum_{v \neq \hat{v}, v \in V} f_v(s_i)]$$
$$= f_{\hat{v}}(s) - f_{\hat{v}}(s_i).$$

where $\hat{v}$ is the sub-problem including locus i. It is clear that fitness differences depend only on the linkage set $\hat{v}$ and independent on loci $j \notin \hat{v}$. Therefore, we can obtain bias in sub-populations classified according to fitness differences and detecting such bias by minimizing the entropy measure we can learn the linkage set for locus i.

Our greedy search shown in Fig. 2 sometimes can not find k bit dependencies, because in some problems entropies for lower order linkage sets shows randomness even if those for higher order sets distribute unevenly. However, it is clear that strings can be divided into at least two sub-populations and there should be 2^{k-1} schemata for loci $\hat{v}$ in these sub-populations. On the other hand, loci $j \notin \hat{v}$ distribute perfectly random and for any k-bit combination of such loci, there should be all possible schemata (2^k schemata). For example, for $\hat{v}$ of 3-bit problem, a sub-population have schemata$\{111, 100, 010, 001\}$. In this sub-population, all order 1 and 2 loci have all possible schemata and only the order 3 loci has half of all. Such dependency can not be found by the greedy search. But if more sophisticated method is applied, it is not impossible. The refinement should require larger computation cost, but is worthy of consideration when fitness evaluation of a problem takes huge time.

4 Population Sizing

In this section, we calculate the number of strings required to detect correct dependencies by the D^5. This consists of two stages: sub-population sizing and overall population sizing. The above case defines the size enough to distinguish biased distribution of dependent loci and random distribution of independent loci in a sub-population. The overall population size must ensure the sufficient sub-population size when it is divided.

We calculate sub-population size and then define overall population size. But, first of all, we should simplify the problem to make population sizing easy.

4.1 Simplification

Sub-population should have enough number of strings to distinguish biased distribution of dependent loci and random distribution.

The level of bias changes with each fitness landscape of a sub-problem. Some of them give strong bias, others give weak bias. The strong bias makes distinction easy, while the weak bias makes it difficult. The former case means that the sub-population has a few unique schemata. Therefore, the D^5 can exploit a small part of original population. In the later case, the sub-population should have more schemata and it can use a larger part of the population. To make calculation easy, we consider the first case only. The resulting population size should not be

upper bounds or precise predictions of population size, as a consequence of this simplicity. But it will help to understand how the number of strings grows as string length gets longer.

As mentioned in section 2, we consider additively decomposable functions only. In this type of functions, there are 2^k schemata for each sub function $f_v(s)$. One bit perturbation gives 2^k schemata changes such as

$$\begin{aligned} df_1(\underline{0}00...) &= f(\underline{0}00...) - f(\underline{1}00...) \\ df_1(\underline{0}01...) &= f(\underline{0}01...) - f(\underline{1}01...) \\ df_1(\underline{0}10...) &= f(\underline{0}10...) - f(\underline{1}10...) \\ &\cdots\cdots\cdots\cdots \\ df_1(\underline{1}11...) &= f(\underline{1}11...) - f(\underline{0}11...) \end{aligned}$$

It is clear that $df_i(s) = -df_i(s_i)$. For example,

$$df_1(\underline{0}00) = f(\underline{0}00...) - f(\underline{1}00...), \quad df_1(\underline{1}00) = f(\underline{1}00...) - f(\underline{0}00...).$$

Then

$$df_1(\underline{0}00) = f(\underline{0}00...) - f(\underline{1}00...) = -df_1(\underline{1}00).$$

We denote fitness difference of h-th schema by perturbation of i-th loci as df_i^h $(i = 1, 2, \cdots, l. \quad h = 1, 2, \cdots, 2^k)$.

The number of unique fitness differences by the perturbations in locus i varies from 2 to 2^k. The upper bound, 2^k, comes in the case where fitness changes of all schemata differ:

$$df_i^h \neq df_i^{h'} \quad (\forall h \neq h') \tag{7}$$

The lower bound, 2, takes place if the amount of fitness increase and decrease are always same:

$$|df_i^h| = |df_i^{h'}| \quad (\forall (h, h')) \tag{8}$$

In order to make calculation easy, we consider the first case. In this case, all 2^k schemata should be classified into different classes. One class has one schema for a linkage set. Loci in the linkage set of all strings in the class have same value. The number of strings must be enough to ensure that no single unlinked locus takes a same value in the class. To this end all classes does not have to have exactly one schema, but at least one of them must have exactly one schema. Therefore, we relax the equation (7) as follows:

$$^{\exists} df_i^{h'} \in \{df_i^1, \cdots df_i^{2^k}\} \quad \text{that satisfies } df_i^{h'} \neq df_i^h \ (h \neq h') \tag{9}$$

If this condition is satisfied, the sub-population of $df_i^{h'}$ has only one unique schema. The entropy of the set of loci that depend on locus i should be zero. On the other hand, the entropy of the set of loci that do not depend on locus i is close to the number of loci in the set. Therefore, if there are enough strings in the sub-population, it is easy to identify linkage set correctly.

Despite such simplicity of linkage identification over a sub-population, whole population size for the function that satisfies the condition (9) should be large. The reason comes from the fact that we can exploit only a small part of whole population for the sub-population of $df_i^{h'}$ because h'-th schema should have $n/2^k$ copies where n is whole population size.

If fitness function does not satisfy (9), by contrast, there are two or more schemata in sub-populations. Consequently, entropies in the sub-populations should be larger than zero. However, they should be still smaller than that for random distributions. For example, the lower bound of fitness differences are 2, as mentioned earlier. In the case, there are 2^{k-1} unique schemata in each sub-population. Then the entropy should be less or equal to $k-1$. The signal difference of entropy increase from the above case, but the half part of original population can be used.

The precise population size for the D^5 is defined from both the string utilization ratio of original population and the signal difference of entropy between $\hat{v}$ and v. Both of those differ with respect to fitness functions. In the followings, to make population sizing easy, we consider the extremely biased case, the functions that satisfy (9) only. Although it does not give the precise population size, it at least should give how it grows with problem size.

After theoretical population sizing, we perform experiments with some classes of problems including the problem that does not satisfy the condition (9).

4.2 Sub-population Sizing

Let $C_{df_i^{h'}}$ a sub-population of $df_i^{h'}$ in (9), let n_1 size of the $C_{df_i^{h'}}$.

Locus $j \notin \hat{v}$ must have less certain distribution than locus $j \in \hat{v}$ to detect correct dependency for locus i.

If (9) is satisfied for additively decomposable functions (1), then there is only one unique schema of $\hat{v}$ in sub-population $C_{df_i^{h'}}$. Therefore, each locus of $\hat{v}$ takes the same gene value in the sub-population.

On the other hand, loci in $V - \{\hat{v}\}$ take 0 or 1 at random. If a locus $j \notin \hat{v}$ takes the same gene value in $C_{df_i^{h'}}$ accidentally, the dependency detection will fail. Therefore, we should employ enough sub-population size n_1 to avoid such undesirable coincidence for all $j \notin \hat{v}$.

The probability that every gene value of a locus $j \notin \hat{v}$ is same is

$$\left(\frac{1}{2}\right)^{n_1}. \tag{10}$$

The probability that it does not occur in all loci $j \in V - \{\hat{v}\}$ is

$$\left(1 - \left(\frac{1}{2}\right)^{n_1}\right)^{l-k}. \tag{11}$$

The probability P_1 that the previous condition hold true for all dependency detections for loci $i = 1, 2, \cdots, l$ is

$$P_1 = \left(\left(1 - \left(\frac{1}{2}\right)^{n_1}\right)^{l-k}\right)^{l} = \left(1 - \left(\frac{1}{2}\right)^{n_1}\right)^{l(l-k)}. \tag{12}$$

Randomly generated gene value in $j \notin \hat{v}$ can have same value in size n_1 sub-population $C_{df_i^{h'}}$ with probability $\left(\frac{1}{2}\right)^{n_1}$ and it can not occur with probability $\left(1 - \left(\frac{1}{2}\right)^{n_1}\right)$. This hold true for all $j \notin \hat{v}$ with probability $\left(1 - \left(\frac{1}{2}\right)^{n_1}\right)^{l-k}$ because $|\hat{v}| = k$ and then $|V - \hat{v}| = l - k$. Therefore the probability that we can avoid the undesirable biases for all $j \in \hat{v}$ in all l times perturbations is equation (12).

Rewriting (12), we can obtain sub-population size for expected success ratio P_1

$$n_1 = -\log(1 - P_1^{\frac{1}{l(l-k)}}). \tag{13}$$

Therefore, if an appropriate distribution is required with probability P_1, then sub-population must have more than $-\log(1 - P_1^{\frac{1}{l(l-k)}})$ strings.

The resulting sub-population size n_1 is also used as threshold for the estimation phase in the D^5. If a sub-populations has less than n_1 strings, it should not be used for dependency detection.

4.3 Overall Population Sizing

Now, we consider overall population size, n. It must be enough to obtain appropriate sub-population size, n_1. From (9), sub-population $C_{df_i^{h'}}$ has only one schema of possible 2^k schemata in $\hat{v}$. Because the original population is initialized by random coin toss, the distribution of the schema in the original population is the binomial distribution with the mean $\frac{n}{2^k}$ and the variance $n\frac{1}{2^k}\left(1 - \frac{1}{2^k}\right)$.

Therefore, the lower bound of expected sub-population size n_1 for large n is estimated as follows:

$$n_1 \geq \frac{n}{2^k} - \sqrt{n\frac{1}{2^k}\left(1 - \frac{1}{2^k}\right)} \tag{14}$$

Let $p = 1/2^k$ and $q = 1 - p$, we rewrite above equation as

$$n_1 \leq np - \sqrt{npq}. \tag{15}$$

Rewriting the inequality, required population size n is as follows:

$$n \geq \frac{(2n_1p + pq) + \sqrt{(2n_1p + pq)^2 - 4p^2n_1^2}}{2p^2}. \tag{16}$$

$$= \frac{1}{p}n_1 + \frac{q}{2p} + \sqrt{\frac{q}{p}n_1 + \frac{q^2}{4p^2}}. \tag{17}$$

Because p is constant if k is constant, the term $\frac{1}{p}n_1$ is dominate for population sizing, thus

$$n = O\left(\frac{1}{p}n_1\right) \tag{18}$$

$$= O(-2^k \log(1 - P_1^{\frac{1}{l(l-k)}})). \tag{19}$$

Using L'Hopital's rule,

$$\lim_{l\to\infty} \frac{1 - P_1^{1/l^2}}{1/l^2} = 1. \tag{20}$$

$$\lim_{l\to\infty} \frac{\ln(1 - P_1^{1/l^2})}{\ln l^2} = \lim_{l\to\infty} \frac{\ln(1 - P_1^{1/l^2})}{2\ln l} = \lim_{l\to\infty} \frac{-P_1^{1/l^2} \ln P_1}{(1 - P_1^{1/l^2})/(1/l^2))}. \tag{21}$$

The limit of the denominator is 1 from the equation (20), the limit of the numerator is $-\ln P_1$. Thus,

$$\lim_{l\to\infty} \frac{\ln(1 - P_1^{1/l^2})}{2\ln l} = -\ln P_1 \tag{22}$$

Therefore, approximating $\log(1 - P_1^{\frac{1}{l(l-k)}})$ to $\log(1 - P_1^{\frac{1}{l^2}})$

$$O(-2^k \log(1 - P_1^{\frac{1}{l(l-k)}})) \approx O(2^k \log l) \tag{23}$$

for large l.

4.4 Overall Complexity

In this section, we show the number of fitness evaluations for the D^5. Optimization using it consists of the dependency detection stage and the building block combination stage. However, if problem structure is revealed, the following evolution should be success using relatively small cost. Therefore, we consider computation cost for dependency detection is approximately equal to overall computation cost.

In dependency detection by the D^5, we should know original fitness of all strings and those after perturbations in all l loci of all strings. Therefore, the number of evaluations required to obtain appropriate linkage sets is $nl+n$ where l is string length and n is population size. substituting (16), the number of evaluations is

$$nl + n = \frac{(2n_1p + pq) + \sqrt{(2n_1p + pq)^2 - 4p^2n_1^2}}{2p^2}(l + 1). \tag{24}$$

If the order of problem k is fixed, then $p = 1/2^k$ and $q = 1 - p$ are also fixed. From the (16) and (24), the number of evaluations is roughly

$$O(n_1 l) = O(-l \log(1 - P_1^{\frac{1}{l(l-k)}})) \tag{25}$$

for string length l.

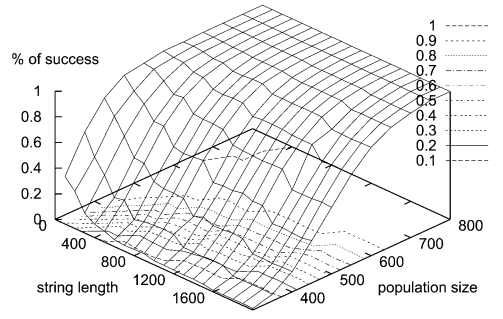

Fig. 4. Accuracy of linkage identification for trap function for various population size and string length

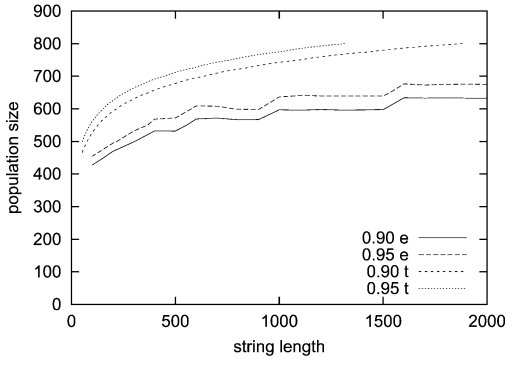

Fig. 5. Contours of fig. 4 and theoretical estimation in equation (16). In index, numbers like 0.95 or 0.90 mean contour levels, e or t means experimental result or theoretical result respectively

5 Experiments: Population Size, String Length and Success Ratio

In this section, we compare theoretical estimations in section 4 and experimental results.

Trap Function

Experiments in this section are performed on a deceptive trap function as follows:

$$f(s) = \sum_{i=1}^{m} \mathrm{trap}(u_i) \tag{26}$$

$$\mathrm{trap}(u_i) = \begin{cases} k & (u = k) \\ k - 1 - u & (\text{otherwise}) \end{cases} \tag{27}$$

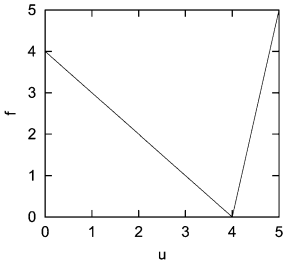

Fig. 6. Example of Trap Function

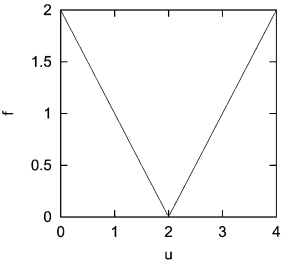

Fig. 7. Example of Valley Function

where m is the number of sub-functions and k is the order of a sub-function, u_i is the number of ones in each k-bit sub-string. This sub-function is called trap function.

Fig.6 shows the function. The x-axis of the figure shows the number of ones in a sub-string and y-axis is the contribution of each sub-string. The left side of the figure shows the contribution of all zeros sub-string and the right side shows that of all ones. From this function, it is clear that the fitness difference for 1-bit perturbation takes one of $\{k, 1, -1, -k\}$. Only all ones gives $-k$ (when $u = k$ to $k - 1$ for perturbation , $11111 \rightarrow 01111$) and only sub-strings to be all ones by the perturbation gives k (when $u = k - 1$ to k for perturbation , $01111 \rightarrow 11111$). All the other 2^{k-1} sub-strings give -1 or 1 by $0 \rightarrow 1$ or $1 \rightarrow 0$ respectively. This function satisfies the assumption 1 because there are two schemata, all ones and that to be all ones, whose fitness differences differ from all the other fitness differences.

In these experiments, we try various string length $l = k \times m$ and population size n. The order of problem is fixed to $k = 5$. We record percent of linkage correctly identified for several (l, n) pairs. We perform 10 runs for each (l, n) pair and average success ratio of linkage identification.

Figure 4 shows the experimental result of accuracy of linkage identification for 5-bit trap function in various population size and string length. Figure 5 shows contours of the accuracy of experimental results and theoretical estimations in equation (16).

Because our theoretical population sizing is conservative – we assume one unique fitness difference but there are two unique fitness differences in the trap function –, the experimental result can archive a certain success ratio with smaller number of strings than the theoretical result. However traces of contour in experiments are follows that in experiment very well.

Valley Function

In this experiment, we employ a test function which does not satisfy (9). The function is defined as follows:

$$f(s) = \sum_{i=1}^{m} \text{valley}(u_i) \tag{28}$$

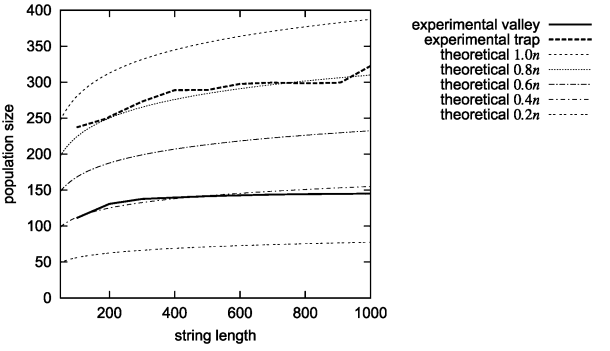

Fig. 8. D^5 for 4-bit valley function and 4-bit trap function. The lines mean threshold sizes for linkage identification with probability 0.95 for each string length. For reference, $c \times n$ where n c-s are constants and vary from 0.2 to 1.0 are added

$$\text{valley}(u) = |u - \frac{k}{2}| \tag{29}$$

where k is the order of a sub-function, m is the number of sub-functions, and u is the number of ones in each k-bit sub-string. Figure 7 shows this function. The valley function has no peak or needle like the optimal solution of the trap function. For all perturbations it gives only two types of fitness difference $\{-1, 1\}$ if k is even.

We fix k to 4 and try various (l, n) pairs to record ratio of correctly identified linkage sets. We perform 10 runs for each (l, n) pair and average success ratio of linkage identification.

Figure 8 shows contours of the accuracy of experimental results and theoretical estimations in equation (16). We also perform 4-bit trap function for comparison. And for reference, $c \times n$ where c-s are constants and vary from 0.2 to 1.0 are added. In this equation, n is the theoretical estimation and $n = -l \log(1 - 0.95^{1/l(l-4)})$.

If many schemata have a same fitness difference, then all of them are classified into one sub-population. The entropy of linkage set in such sub-population is larger and closer to the entropy of random set of loci than the entropy of linkage set of a function which has various fitness differences like a trap function. On the other hand, because original population is divided into a few groups in this kind of problems, each sub-population size should be large and enough to detect the small signal difference of entropy.

For the 4-bit functions, there are $n_1^t = n^t/2^4 = n^t/16$ strings for the trap function and there are $n_1^v = n^v/2$ strings for the valley function where n^t is population size for trap function and n^v is population size for valley function. Because $n^v \approx n^t/2$ from the experiment, the sub-population size in valley function is $n_1^v = n^t/4 = 4n_1^t$. Sub-population sizes n_1^v and n_1^t are defined from the signal difference of entropy and population sizes n^v and n^t are defined to ensure sub-population sizes.

Although the number of strings for the valley function is smaller than for the trap function, the slope for the valley function is similar to that for the trap function. In fact, the population size for the valley function is similar to the $0.4 \times n$ and that for the trap function is similar to the $0.8 \times n$. These results show that the theoretical population size guides population sizings for problems which do not satisfy the condition (9).

6 Comparisons with Other Methods

In this section, we compare population sizings in existing literatures of GAs, PMs and EDAs. Depending on the strategies of each algorithm, their way of sizing differs. Please not that these population sizings do not address the same problem and the same purpose. However, they have some things in common and other things in contrast.

6.1 Population Sizing of Simple GAs

Various efforts have been focused on sizing population of genetic algorithms. One of the most accurate population size was calculated by Harik et al. [HCPGM97].

They estimated population size considering an initial supply of building blocks and a good decision making between competing building blocks using the gambler's ruin model. Resulting population size n that is enough for an optimal solution to take over population after several generations is

$$n = -2^{k-1} \ln(\alpha) \frac{\sigma_{bb} \sqrt{\pi(m-1)}}{d} \tag{30}$$

where k is the order of a building block, α is the probability of failure convergence in a sub-problem, σ_{bb} is the standard deviation (the square root of variance) of fitness of the building block, m is the number of building blocks (sub-problems) in a string, and d is the difference between the mean fitness of the best and the second building blocks. The term 2^{k-1} is required for the initial supply of building blocks and the other terms are for the decision making. The last term shows that if the population size increases as the average variance of the building blocks increases, as the problem size is grows and as the signal difference decreases.

If we assume that string length is approximately proportional to the number of building blocks, we obtain following result from equation (30):

$$n = O(2^{k-1} \sqrt{l}) \tag{31}$$

In their analysis, they assumed tight linkage and building block disruption was not considered. If such tight linkage is not ensured, SGA performs as random search and needs $O(l^2)$ strings for the worst case. In addition, some approximations are used in their calculation.

6.2 Population Sizing of PMs

For the LINC, the number of strings required to obtain correct linkage set was calculated by Munetomo et al [MG99]. The LINC identifies linkage by detecting the second order nonlinearity. It assumes that nonlinearity must exist within loci to form a building block.

This population sizing is differ from the population sizing of simple GAs because they do not concern the number of strings for optimal population convergence but for correct linkage identifications. However, if correct linkage sets are obtained it becomes easy to combine building blocks to find an optimal solution.

Their population sizing is based on the supply of building blocks in a linkage set because they assume that there is at least one schema which violates linear condition along the perturbations for the pair of loci belonging a same sub-problem. They showed that if there are n strings then correct linkage sets is obtained with probability P as follows:

$$P = 1 - \left(1 - \left(\frac{1}{2^k}\right)\right)^n \tag{32}$$

where k is the order of a sub-problem. Therefore the population size required for a certain success probability P is

$$n = \frac{\log(1-P)}{\log(1-1/2^k)} \simeq -2^k \log(1-P). \tag{33}$$

From equation (33), the population size for the LINC depends only on the order of sub-problem.

Munetomo et al [MG99] calculate the number of strings which is enough to obtain pairs of loci that are linked with a given probability, P. Whereas, the population size in the D^5 is sized enough to obtain all sets of loci that linked. In addition, Heckendorn et al [HW03] define the number of strings to find all pairs of loci that linked with a global probability of success. The population size of their algorithm (the number of iterations in their algorithm) is $-2^k \ln(1-\delta^{1/J})$ for order-2 linkage detection where δ is the probability of success and J is the number of order-2 relationships between loci (hyperedges). For the non-overlapping additively decomposable functions composed of order-k sub functions, $J = l/k \times_k C_2 = l(k-1)/2$. Then the population size is $-2^k \ln(1-\delta^{2/l(k-1)})$.

6.3 Population Sizing of EDAs

As an example of population sizing in EDAs, we show the population sizing for the BOA calculated by Pelikan et al.[PSG02] The BOA [PGCP99] exploits Bayesian network to represent conditional probabilities in order to encode dependency of variables in their models.

For uniformly scaled problem, the most important factor for population sizing is that the BIC metric can distinguish between the appropriate and the

Table 1. Comparison of population size and number of evaluations

method	population size	number of evaluations
D^5	$O(-2^k \log(1 - P_1^{\frac{1}{l(l-k)}}))$	$O(-2^k l \log(1 - P_1^{\frac{1}{l(l-k)}}))$
SGA	$O(2^{k-1}\sqrt{l}) \sim O(2^l)$	$O(2^{k-1}l) \sim O(2^{k-1}l \log l) \sim O(2^{k-1}l^2)$
LINC	$O(2^k)$	$O(2^k l^2)$
BOA	$O(2^k l^{1.05})$	$O(2^k l^{1.55})$

inappropriate dependencies and decision making between to add or to not add an edge from a variance to another variance. From the viewpoint, the required number of strings is

$$n = O(l^{1.05}). \tag{34}$$

This result is obtained using some approximations, however, it matches experimental results. The detail about this equation is available in the literature [PSG02].

Above equations were obtained for two bit dependency, but it can be extended for multiple dependencies as follows:

$$n = O(2^k l^{1.05}) \tag{35}$$

The term 2^k comes from the number of possible schemata and $l^{1.05}$ is the requirement for noise avoidance from contributions of other sub-solutions.

6.4 Discussions

Table 1 shows the comparison of population size in the D^5, SGA, the LINC and the BOA. Please note that the population size of SGA $O(2^{k-1}\sqrt{l})$ is for tight encoding strings. If such tight encoding is not ensured SGA requires an exhaustive search which needs exponential number of strings. In addition, the comparison is not completely fair because the population sizings for the LINC and the D^5 does not concern evolution of population and decision-making during the evolution. However, it should be true that if we know problem structure, then we can make decision easier than without such explicit information of problems.

The required number of evaluations is also shown in Table 1. Again, the numbers of the D^5 and the LINC are only for dependency detection and they need other evaluations for evolution phase but these are not dominant for overall computation cost. The number of evaluations for SGA and the BOA is simple multiply of population size and the number of generations for convergence. The numbers of evaluations for SGA are varied with respect to the selection methods [GD91] and is also an ideal case that tight linkage can be ensured from previous information of problems.

Streeter [Str04] improves the LINC from a traditional algorithm perspective. It uses binary search to detect specific loci j which depend on i. Therefore, it requires $O(2^k l \log l)$ fitness evaluations where $l \log l$ comes from binary search for each locus $i = 1, 2, \cdots, l$ and $O(2^k)$ is population size required to guarantee

a certain success probability. This computation cost is similar to our result, $O(-2^k l \log(1 - P_1^{1/l(l-k)}))$. The main difference is that the log term of the D^5 comes from population size, $O(-2^k \log(1 - P_1^{1/l(l-k)})$, while the Streeter's one comes from binary search.

All of population sizings have the factors from possible number of schemata, 2^k or $2^{k-1} = 2^{-1} \times 2^k$. This number, 2^k, is number of all possible schemata of order k. It should guarantee existence optimal schema which will be produced with probability $1/2^k$. SGA and BOA consider the effect from other sub-functions by $\sqrt{l}$ and $l^{1.05}$ respectively. The D^5 has also the term $\log(1 - P_1^{1/l(l-k)})$ for population sizing. This comes from requirement to avoid undesirable bias for all loci which do not belong a sub-problem including a perturbed locus. This requirement is approximately equal to the requirement that initial population should distribute enough randomly. The LINC and Streeter's algorithm, which are uses perturbations only for their dependency detections, have no term for string length in population sizing.

7 Conclusion

In this paper, we estimate the number of strings required for the D^5 under some assumptions. Estimated population size is $O(-2^k \log(1 - P_1^{\frac{1}{l(l-k)}})$ where l is string length and k is the order of the sub-problem. This result shows the number of strings required to obtain correct linkage sets is defined mainly by the order of sub-problem and the D^5 can be scalable to large problem size. Validity of the population is also verified in experiments. The experimental population sizes follow $c \times -2^k \log(1 - P_1^{\frac{1}{l(l-k)}})$ where c is a constant.

References

[GD91] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In Gregory J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann Publishers, 1991.

[GKD89] David E. Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):415–444, 1989.

[HCPGM97] Georges Harik, Erick Cantú-Paz, David E. Goldberg, and Brad L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pages 7–12, 1997.

[HW03] Robert B. Heckendorn and Alden H. Wright. Efficient linkage discovery by limited probing. In *Proceedings of the 2003 Genetic and Evolutionary Computation Conference*, pages 1003–1014. Morgan Kaufmann Publishers, 12–16 July 2003.

[Kar95] H. Kargupta. SEARCH, polynomial complexity, and the fast messy genetic algorithm. Technical Report 95008, University of Illinois at Urbana-Champaign, Urbana, IL, October 1995.

[LGP00] Fernando G. Lobo, David E. Goldberg, and Martin Pelikan. Time complexity of genetic algorithms on exponentially scaled problems. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference*, pages 151–158. Morgan Kaufmann Publishers, 10-12 July 2000.

[MG99] Masaharu Munetomo and David E. Goldberg. Identifying linkage groups by nonlinearity/non-monotonicity detection. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, pages 433–440, 7 1999.

[PGCP99] Martin Pelikan, David E. Goldberg, and Erick Cantú-Paz. BOA: The bayesian optimization algorithm. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, pages 525–532. Morgan Kaufmann Publishers, 1999.

[PSG02] Martin Pelikan, Kumara Sastry, and David E. Goldberg. Scalability of the bayesian optimization algorithm. *International Journal of Approximate Reasoning*, 31(3):221–258, 2002.

[Str04] Matthew J. Streeter. Upper bounds on the time and space complexity of optimizing additively separable functions. In *Proceedings of the 2004 Genetic and Evolutionary Computation Conference*, pages 186–197, 2004.

[TGP98] Dirk Thierens, David E. Goldberg, and Ângela G. Pereira. Domino convergence, drift and the temporalsalience structure of problems. In *Proceedings of the IEEE International Conference of Evolutionary Computation*, pages 535–540, 1998.

[TMA04] Miwako Tsuji, Masaharu Munetomo, and Kiyoshi Akamae. Modeling dependencies of loci with string classification according to fitness differences. In *Proceedings of the 2004 Genetic and Evolutionary Computation Conference*, pages 246–257, 26-30 June 2004.