A Parallel Genetic Algorithm Based on Linkage Identification

Masaharu Munetomo, Naoya Murao, and Kiyoshi Akama

Hokkaido University, North 11, West 5, Sapporo 060-0811, JAPAN.
{munetomo, naoya.m, akama}@cims.hokudai.ac.jp

Abstract. Linkage identification algorithms identify linkage groups — sets of loci tightly linked — before genetic optimizations for their recombination operators to work effectively and reliably. This paper proposes a parallel genetic algorithm (GA) based on the linkage identification algorithm and shows its effectiveness compared with other conventional parallel GAs such as master-slave and island models. This paper also discusses applicability of the parallel GAs that tries to answer "which method of the parallel GA should be employed to solve a problem?"

1 Introduction

To solve large, complex combinatorial optimization problems, parallelization of optimization algorithms is crucial. Evolutionary computation models such as genetic algorithms (GAs) are considered suitable for parallel computation because they deal with a number of strings that can be processed independently in their fitness evaluations and genetic operators such as mutations and crossovers. Therefore, the parallel GA is an active research area and a number of papers have been published in this area. Typical approaches of the parallel GAs are master-slave models, island models, massively parallel GAs, and so on. Although they succeeded in parallelizing GAs, it does not mean that they succeeded in parallelizing their optimization, in other words, they can reduce time to obtain optimal solutions properly as compared with the number of processors employed. Speedup analysis is essential to discussing efficiency of parallel GAs. Cantu-Paz et. al[1] performed theoretical and empirical analysis on speedup for master-slave models, island models, and so on. The analysis shows the conditions for the models to work effectively; for example, master-slave models work effectively when time to evaluate fitness is much greater than that for communications among processors. In this paper, we first review and discuss current technologies and the conditions that they work effectively.

Discussing efficiency of parallelizations, we need to discuss effectiveness of the GA itself in solving a problem. This is because when a GA (with some operators, selection schemes, etc.) cannot solve a problem essentially, we cannot solve the problem even though parallelized version of the GA is employed. This happens especially when we employ simple GAs to solve difficult problems that have deceptiveness and encoded strings do not have tight linkage. If linkage is loose, building blocks are easily disrupted by simple crossovers and therefore genetic optimizations are easily trapped into deceptive attractors. On the consequence of this, we cannot expect the GA to obtain optimal solutions because it is easily converged to local optima. To solve this problem, linkage identification procedures such as the LINC (Linkage Identification by Nonlinearity Check) [7, 6] and the LIEM (Linkage Identification with Epistasis Measures) have been proposed to identify linkage groups — sets of loci tightly linked to form building blocks. The merit of this approach is not only it can avoid disruption of building blocks but also it can easily be parallelized because its calculation is highly independent. This paper discusses merit of the parallel GA based on the linkage identification compared with conventional parallel GAs.

2 Parallel GAs

A number of papers have been published on parallel GAs. Majority of them are classified into the following categories:

- *master-slave models* that parallelize fitness evaluations
- *island models* that divide a population into subpopulations
- *massively parallel GAs* that place strings on a mesh, etc.
- *hierarchical models* that combine more than two methods

A master-slave model consists of a master that performs GAs and slaves that evaluate fitness values. This model performs fitness evaluations in parallel to enhance its performance. To apply this model, communication overheads should be relatively small compared with time to evaluate fitness values. This is analyzed by Cantu-Paz[1] in his doctoral thesis. In the following analysis based on his thesis, we denote time to evaluate a fitness value as T_f, communication time between processors as T_c, the overall population size as n and the number of processors of the target parallel machine as P. We can easily calculate overall execution time T_p by the P-processor parallel machine as follows:

$$T_p = PT_c + \frac{nT_f}{P}. \tag{1}$$

Speedup factor S is obtained by calculating nT_f/T_p:

$$S = \frac{nT_f}{T_p} = \frac{nT_f}{PT_c + nT_f/P}. \tag{2}$$

This is illustrated in figure 1. This figure shows that parallelization by the master-slave model is effective when $T_f \gg T_c$.

From the above calculations, we obtain the optimal number of processors P^* that maximizes the speedup factor S as follows:

$$P^* = \sqrt{nT_f/T_c}. \tag{3}$$

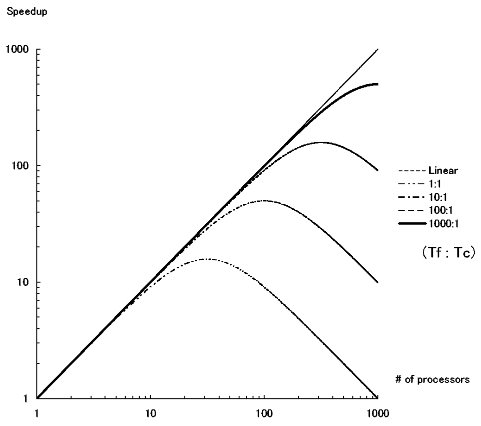

Fig. 1. Speedup by master-slave PGAs

Island models also called subpopulation-based parallel GAs divide a population into subpopulations assigned to processors. Each processor performs a GA for its subpopulation and migrates strings between subpopulations to exchange building blocks. Even though speedup realized by this model is relatively small as illustrated in figure 2, this is considered natural implementation of parallel GAs and may be employed when master-slave models does not work effectively.

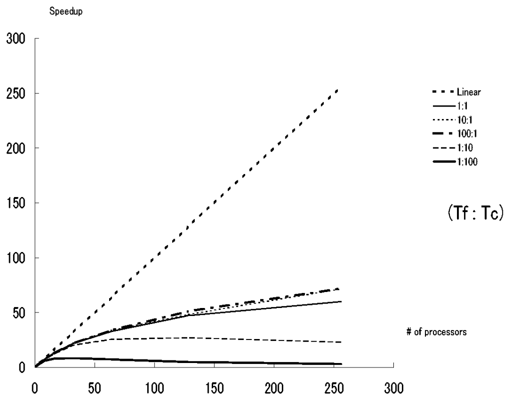

Fig. 2. An Empirical Result of Speedup by an island model PGA

The parallel GAs based on the island model expect that each subpopulation searches different candidates of building blocks to be exchanged among them, however, such favorable situations may not be realized in some problems. For example, when fitness contribution of building blocks to overall fitness is ex-

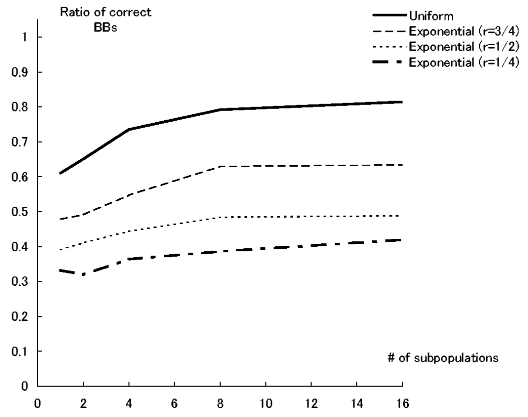

Fig. 3. Ratio of BBs correctly identified by an island model

ponentially decreasing (or increasing), all subpopulations must search the same building block with maximum fitness contribution at first, then search that with the second maximum, and so on. This is reported by Goldberg in discussing continuation operators[4].

Figure 3 shows that subpopulation-based parallel GAs performs well when fitness contribution of building blocks is uniform and their performance gets worse when the contribution becomes exponential. In the figure, we employ the sum of 5-bit trap functions as a test function and we assign a weight w_i to each i-th trap subfunction ($i = 1, 2, \cdots, 20$). In uniform case, $w_i = 1$ for all i. In exponential case, $w_1 = 1.0$ and $w_i = r \times w_{i-1}$, that is, the weights are exponentially decreasing. When $r < 1/2$, the sum of all weights for $i = 2, 3, \cdots$ becomes smaller than w_1, the maximum weight value. This means that fitness contribution by the first BB exceeds those for the sum of other BBs, and therefore at first only the BB with the maximum weight should be found in all the subpopulations. This is because diversity cannot be ensured even though a number of subpopulations are employed in the island models.

3 Linkage Identification

A series of linkage identification techniques have been proposed to ensure tight linkage among loci to form building blocks. The Linkage Identification by Nonlinearity Check (LINC)[7,6] identifies linkage groups by introducing bit-wise perturbations for pairs of loci to detect nonlinear interactions of fitness changes. In the LINC, we calculate the following values for each pair of loci (i, j) and for all the strings s in a randomly initialized population that has $O(c2^k)$ strings where c is a constant and k is the maximum order of linkages:

$$\begin{aligned} \Delta f_i(s) = &\quad f(..\bar{s_i}.....) - f(..s_i.....) \\ \Delta f_j(s) = &\quad f(.....\bar{s_j}..) - f(.....s_j..) \end{aligned}$$

$$\Delta f_{ij}(s) = f(..\bar{s_i}.\bar{s_j}..) - f(..s_i.s_j...), \tag{4}$$

where $f(s)$ is a fitness functions of a string s and $\bar{s_i} = 1 - s_i$ ($0 \to 1$ or $1 \to 0$) stands for a bitwise perturbation at the i-th locus. We can consider the following two cases:

1. If $\Delta f_{ij}(s) \neq \Delta f_i(s) + \Delta f_j(s)$, then s_i and s_j are surely members of a linkage group, so we add i to the linkage group of locus j and add j to the linkage group 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 a member of a linkage group, or they are linked but linearity exists in the current context. We do nothing in this case.

The LINC checks the above conditions for all the string s in a population. When the nonlinear condition $\Delta f_{ij}(s) \neq \Delta f_i(s) + \Delta f_j(s)$ is satisfied for at least one string s, the pair (i, j) should be considered to be tightly linked. This is because a pair of loci can be optimized independently when linearity is detected for all the string. The detection of nonlinearity is the key idea of the LINC.

The Linkage Identification with Epistasis Measures (LIEM) extends the idea of the LINC by introducing an *epistasis measure* that represents *tightness of linkage* based on the nonlinearity conditions of the LINC. The epistasis measure of the LIEM is defined as follows:

$$e_{ij} = \max_{s \in P} |\Delta f_{ij}(s) - (\Delta f_i(s) + \Delta f_j(s))|, \tag{5}$$

where $\Delta f_i(s)$, $\Delta f_j(s)$, $\Delta f_{ij}(s)$ are the same as those in equations (4).

This measure shows the maximum distance from where the LINC condition is satisfied, therefore, this shows a *degree of dissatisfaction* of the condition. When the measure for a pair of loci (i, j) is equal to zero, the pair should be separable from the LINC conditions. By introducing the epistasis measure, we can relax strict condition of the LINC and can introduce a clear definition of *tightness of linkage* for pairs of loci.

Figure 4 shows the algorithm of the LIEM. First, we randomly initialize a large enough initial population. From theoretical investigations for the LINC, we need to have $O(k2^k)$ strings in a population to identify linkage groups of order k. Secondly, we calculate an epistasis measure e_{ij} for each pair of loci (i, j) by applying perturbations to the pair of loci. After the calculations, the measures are sorted by descendent order. Linkage groups are generated by picking up loci from the first to the k-th sorted measures.

Although the algorithm of the LIEM seems too simple to identify accurate linkage groups, it can identify 100% correct linkage groups for problems with linear combinations of nonlinear subfunctions such as the sum of trap functions even though their bit positions are randomly encoded, and it also achieves accurate identifications for quasi-linear or weak nonlinear combinations of nonlinear subfunctions such as weak nonlinear functions of the sum of trap functions[5]. This is because the LIEM differentes strong nonlinearity from linear or weak nonlinear interactions.

```
algorithm LIEM

N = c*2^difficulty;
P = initialize N strings;
/*  Calculate epistasis measure e[i][j]  */
for i = 0 to l-1
   for j = 0 to l-1
      e[i][j] = 0;
      if i != j then
         for each s in P
            s' = perturb(s, i);
            f1 = fitness(s') - fitness(s);
            s'' = perturb(s, j);
            f2 = fitness(s'') - fitness(s);
            s''' = perturb(s', j);
            f12 = fitness(s''') - fitness(s);
            ep[s] = |f12 - (f1+f2)|;
            if(ep[s] > e[i][j]) then e[i][j] = ep[s];
         endfor
      endif
   endfor
endfor
/*  Generate linkage group l[i][k]
    where k = 0, 1,..., difficulty-1  */
for i = 0 to l-1
   for j = 0 to l-1
      id[j] = j;
   endfor
   sort e[i][j] with j by descendent order;
  /* select linkages */
   for k = 0 to difficulty-1
      if(e[i][k] > epsilon) l[i][k] = id[i][k];
      else break;
   endfor
endfor
```

Fig. 4. The Linkage Identification with Epistasis Measure (LIEM)

4 A Parallel GA Based on Linkage Identification

In this paper, we propose a parallel GA based on the linkage identification technique. The linkage identification algorithms such as the LINC and the LIEM are easy to parallelize, because their calculations of epistasis measures are considered highly independent. Figure 5 illustrates an execution flowchart of our method.

To parallelize linkage identifications, we assign calculations of epistasis measures to processors in a parallel computer. First, a master processor randomly initializes a population of $N = O(k2^k)$ strings and broadcasts them to all the slave processors. As a consequence of this, all the processors have the same population and they can calculate their assigned epistasis measures. Computational cost for calculating epistasis measures for all the pair of loci is $O(l^2)$ (more precisely, we need to calculate $(l^2 - l)/2$ epistasis values). In order to calculate one epistasis measure, we need to evaluate $3N$ fitness values because calculations of $\Delta f_i(s)$, $\Delta f_j(s)$, $\Delta f_{ij}(s)$ are necessary for each s in the population. After cal-

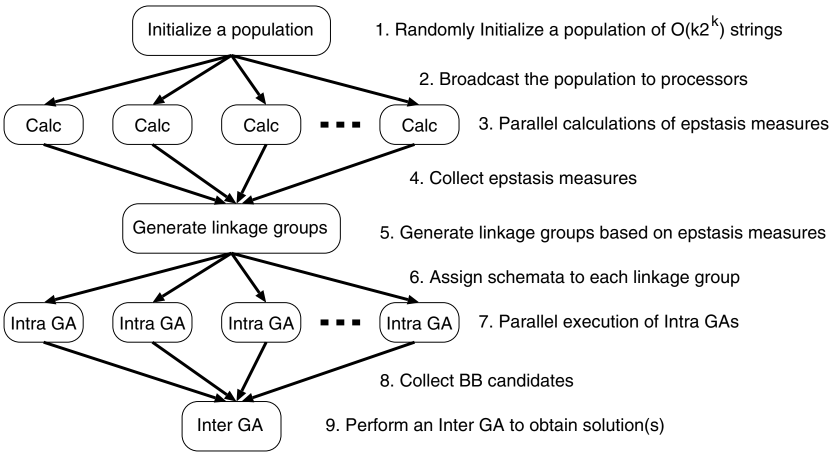
Initialize a population
1. Randomly Initialize a population of O(k2^k) strings
2. Broadcast the population to processors
Calc
Calc
Calc
Calc
3. Parallel calculations of epstasis measures
4. Collect epstasis measures
Generate linkage groups
5. Generate linkage groups based on epstasis measures
6. Assign schemata to each linkage group
Intra GA
Intra GA
Intra GA
Intra GA
7. Parallel execution of Intra GAs
8. Collect BB candidates
Inter GA
9. Perform an Inter GA to obtain solution(s)

Fig. 5. An overview of PGA based on linkage identification

culations of the epistasis measures, they are collected to the master processor which generates linkage groups based on them. This is essentially the same as the serial version of the LIEM.

When we have P processors in a parallel computer, overall computation time T_p for the parallel linkage identification becomes as follows:

$$T_p = \frac{3NT_f(l^2-l)/2}{P} + T_cN + T_c'(l^2-l), \tag{6}$$

where T_f and T_c are the same as in equation (1) and T_c' is the time to send/receive one epistasis measure ($T_c' < T_c$). Similar to the master-slave model, we can calculate speedup factor S for the proposed algorithm as follows:

$$S = (3T_fN(l^2-l)/2)/T_p = \frac{3T_fN(l^2-l)/2}{\frac{3T_fN(l^2-l)/2}{P} + T_cN + T_c'(l^2-l)/2}. \tag{7}$$

We can expect effective parallelization with this algorithm because communication time to send an epistasis measure is much smaller than that for calculating it.

After the generation of linkage groups, we perform Intra GAs in slave processors to search candidates of building blocks and an Inter GA in the master processor to mix and test the candidates to find optimal solutions. The Intra GAs and the Inter GA were originally introduced in a report that proposes a GA based on the LINC[6]. To perform the Intra GAs, we divide initial strings into schemata based on the obtained linkage groups. Figure 6 illustrates this decomposition process.

The Intra GA applies ranking selections, uniform crossovers and simple mutations to search candidates of building blocks in the linkage group. To evaluate schemata, competitive template in messy GA[3] is employed. After the Intra

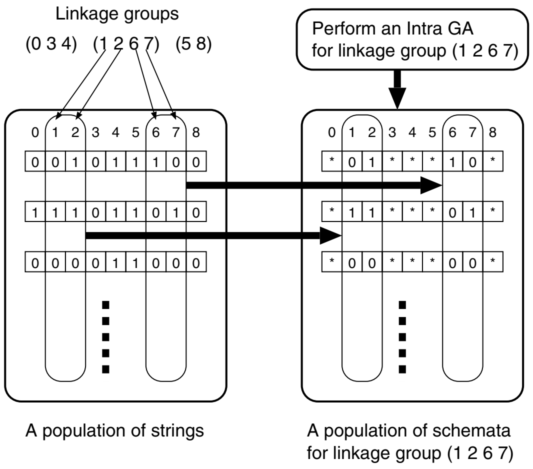

Fig. 6. Division of strings into schemata based on linkage groups

GAs, we select a limited number of well-performed schemata as building block candidates in each linkage group.

The Inter GA processes obtained building block candidates by applying crossovers based on the linkage groups and ranking selections repeatedly. Figure 7 shows crossover operator of the Inter GA. Building block candidates are mixed by this operator and their combinations are tested through selections in order to obtain optimal solutions.

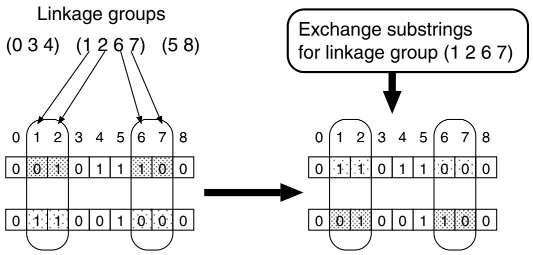

Fig. 7. Crossover operator of the Inter GA

5 Empirical Results

We perform numerical experiments for the proposed parallel GA based linkage identification. We employ the following weighted sum of trap subfunctions as a test function.

$$f(s) = \sum_{i=1}^{L} w_i f_i(u_i). \quad (8)$$

where $w_i > 0$ is the weight of subfunction f_i of unitation u_i (the number of 1's in the K-bit i-th substring of s) defined as follows:

$$f_i(u_i) = \begin{cases} K - u_i & \text{if } 0 \leq u_i \leq K - 1 \\ K & \text{if } u_i = K \end{cases} \quad (9)$$

In the following experiments, we consider the following two cases:

Uniform : All weights have the same value: $w_i = 1$ for all i.
Exponential : Weights are exponentially decreasing: $w_1 = 1$, $w_i = rw_{i-1}$ ($i = 2, 3, \cdots, L$, $0 < r < 1$.)

Fig 8 shows the result on speedup by the parallel GA based on linkage identification. In this experiment, we employ uniform case of the test function. For exponential functions, similar result should be obtained because linkage identification algorithms do not depend on scaling of the fitness function. In this experiment, we change ratio of time for fitness evaluations (T_f) and that for communications (T_c).

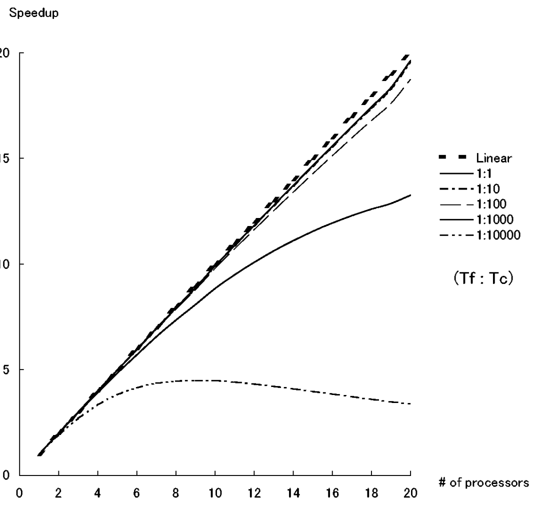

Fig. 8. Speedup by a PGA with linkage identification

This result shows that the PGA with linkage identification achieves near linear speedup except when communication overheads are extraordinary large compared with those for fitness evaluations. (Note that ratios of T_f and T_c in this figure are different from those in the figure 1.)

In figure 9, we compare overall performances of an island model PGA to that with parallel linkage identification for the exponential test function. In the figure, the x-axis shows problem size (total length of the string = LK) and the y-axis shows overall time to obtain optimal solutions ($\times T_f$). In this experiment, we assume the parallel machine has 8 processors and we optimize population size for each model and each string length.

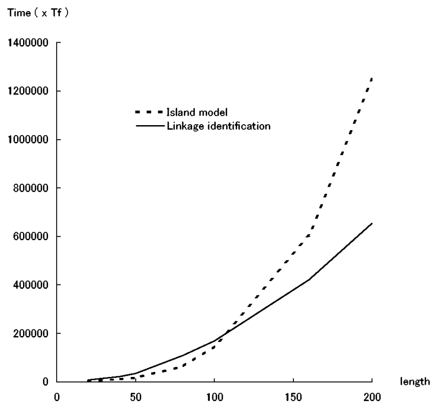

Fig. 9. Problem size vs. time (exponential function)

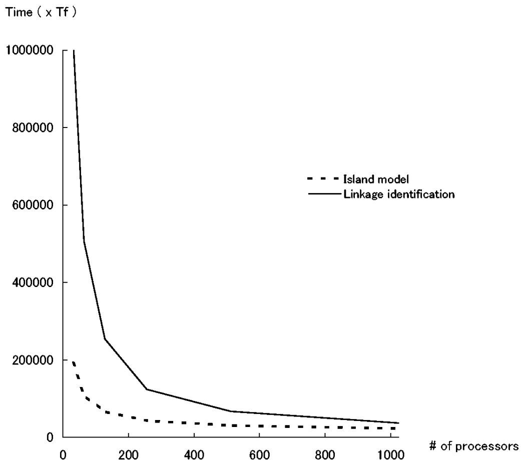

Fig. 10. # of processors vs. time (uniform function)

This figure shows that PGA with linkage identification can find optimal solutions with less computational cost compared with that based on an island model when string length is larger than around a hundred. This is because an island model cannot search effectively a variety of building blocks in their subpopulations, on the other hand, the PGA based on linkage identification can search building block candidates separately in each linkage group. Another reason for the performance difference is that an island model needs larger population size when signal difference of subfunctions becomes small. This is easily understood from the equation of population sizing calculated by Goldberg et. al[2] as follows:

$$n = 2c\kappa \frac{\sigma_M^2}{d^2}, \tag{10}$$

where n is the necessary initial population size, c is a constant determined by sampling noise, κ is the cardinarity of competing schemata, σ_M is the standard deviation of the fitness distribution, and d is the signal difference of the fitness between the best and the second best schemata.

When signal difference of fitness d in the equation becomes smaller, the population size n necessary to obtain optimal solutions becomes larger. In the exponential case, when string length becomes longer, minimum difference of the fitness decreases exponentially, therefore, an island model PGA needs much larger size of population. In contrast, the PGA with linkage identification do not depend on such fitness scaling effect and its necessary population size is constant with respect to signal differences.

Figure 10 shows time to obtain optimal solutions for an island model PGA and that with linkage identification for the uniform case. In this figure, the x-axis shows the number of processors employed and the y-axis shows the time to obtain optimal solutions. In this experiment, we employ a uniform test function whose string length is 500, and we optimize initial population size for both models.

This figure clearly shows that the island model performs better than the PGA with linkage identification in the uniform test function. This is because an island model works effectively by maintaining variety of building block candidates in each subpopulation. On the other hand, PGA with linkage identification needs relatively large computational overheads to obtain accurate linkage information.

6 Conclusions

In this paper, we propose a PGA based on linkage identification. The PGA achieves quasi-linear speedup except when communication overheads is extremely large compared with that for fitness evaluations. In summary, we can select one of the PGA models to solve optimization problems according to the following guidelines:

- If the problem is difficult to ensure tight linkage in advance with prior knowledge, linkage identifications are necessary which can be parallelized easily.
- If time to evaluate fitness values are much larger than that of communications among processors, the master-slave models work effectively.

– If the fitness function of the problem consists of uniformly weighted subfunctions, island models can maintain diversity of building block candidates and are expected to perform well.
– If the fitness function consists of non-uniform subfunctions whose weights are largely different, parallel linkage identifications are expected to perform better than island models.

Our future works includes applications of the proposed model to real world design problems such as broadband network design problems, and we also plan to compare the PGA with linkage identification with parallelized version of advanced GAs and estimation of distribution algorithms such as the Bayesian optimization algorithm.

References

1. Erick Cantú-Paz. *Designing Efficient and Accurate Parallel Genetic Algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, 1999.
2. D. E. Goldberg. Sizing populations for serial and parallel genetic algorithms. pages 70–79, 1989. (Also TCGA Report 88004).
3. D. E. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms revisited: Studies in mixed size and scale. *Complex Systems*, 4:415–444, 1990.
4. David E. Goldberg. Using time effectively: Genetic-evolutionary algorithms and the continuation problem. Technical Report IlliGAL Report No.99002, University of Illinois at Urbana-Champaign, 1999.
5. Masaharu Munetomo. Linkage identification based on epistasis measures to realize efficient genetic algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002.
6. Masaharu Munetomo and David E. Goldberg. Designing a genetic algorithm using the linkage identification by nonlinearity check. Technical Report IlliGAL Report No.98014, University of Illinois at Urbana-Champaign, 1998.
7. Masaharu Munetomo and David E. Goldberg. Identifying linkage by nonlinearity check. Technical Report IlliGAL Report No.98012, University of Illinois at Urbana-Champaign, 1998.