Empirical Investigations on Parallel Competent Genetic Algorithms

Miwako Tsuji
Hokkaido University
N11W5, Sapporo
060–0811,Japan
m_tsuji@cims.hokudai.ac.jp

Masaharu Munetomo
Hokkaido University
N11W5, Sapporo
060–0811,Japan
munetomo@iic.hokudai.ac.jp

Kiyoshi Akama
Hokkaido University
N11W5, Sapporo
060–0811,Japan
akama@iic.hokudai.ac.jp

ABSTRACT

This paper empirically investigates parallel competent genetic algorithms (cGAs) [4]. cGAs, such as BOA [21], LINC-GA [15], D^5-GA [28], can solve GA-difficult problems by automatically learning problem structure as gene linkage. Parallel implementation of cGAs can reduce computational cost due to the linkage learning and give us problem solving environments for a wide spectrum of real-world problems. Although some parallel cGAs have been proposed [16,18,19], the effect of the parallelizations has not been investigated enough. This paper empirically discusses the applicability and property of parallel cGAs, including a new parallel cGA, parallel D^5-GA.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms

Keywords

Linkage, Parallelization

1. INTRODUCTION

Competent genetic algorithms (cGAs) [4] are designed to solve hard problems quickly, reliably and accurately. To solve a wide spectrum of real-world problems, effective building block (BB) mixing is essential. However, it is sometimes difficult to encode strings to ensure that the traditional one- (or multi-) point crossover mixes BBs properly. Therefore, cGAs must learn linkage — the relationship between variables tightly linked to form a BB. In fact, cGAs calculate fitness differences and/or estimate the distribution of promising strings to learn the linkage [8, 9, 12, 15, 20, 21, 23, 25].

Linkage identification techniques which calculate fitness differences caused by perturbation of variables are called perturbation methods (PMs) and those which estimate the distribution of promising strings are called estimation of distribution algorithms (EDAs).

Recently, parallel cGAs have been developed to reduce new computational costs due to the linkage learnings. For example, PBOA, DBOA, parallel BOA [18, 19] based on BOA [21] and pLINC [16] based on LINC [15] have been proposed. The parallel versions of BOA constructs models in parallel in addition to parallel evaluation of offsprings. LINC performs numerous fitness evaluations to identify linkage and pLINC does these evaluations in parallel. These parallel cGAs have saved computational time in a parallel computation environment.

However, there have been few investigations on the effect of the parallelizations of cGAs [17]. The scalability of cGAs with problem size in a single processor has been realized. Then how the scalability with the number of processors is? In this paper, we empirically discuss the applicability and property of some parallel cGAs. Before the discussion, we add a new parallel cGA called parallel D^5-GA in our investigations. The D^5 (Dependency detection for distribution derived from fitness differences) [28] identifies the linkage in a hybrid manner, by combining linkage identification techniques of PMs and EDAs. To exploit the linkage, D^5-GA employs a crossover method called CDC (Context dependent crossover) [27] which can combine overlapping building blocks.

This paper is organized as follows. We provide a brief review on parallel GAs, linkage, and cGAs in the next section. In section 3, we parallelize D^5-GA in addition to the reviews of D^5 and CDC. In section 4, we perform experiments to discuss parallel cGAs. Finally, we conclude in section 5.

2. BACKGROUND

2.1 Parallel GAs

Because GAs work with a set of candidate solutions rather than a single point, it had been expected that parallel implementations of GAs would reduce the time to find a feasible solution [1,3].

In early days of GAs, simple GAs had been implemented in parallel using master-slave model [2], island model [5] [6], etc. The master-slave model GAs compute fitness in multiple processors and apply other operators in a master processor. In most cases, they provide solutions which are equiv-

```
(1) set $t := 0$, initialize population $P_t$
(2) select promising solutions $P'_t$ from $P_t$
(3) construct a Bayesian network $B$ based on the $P'_t$
(4) sample new solutions $O_t$ from $B$
(5) replace some better strings in $P_t$ with $O_t$
(6) if the termination criterion is not met, let $t := t + 1$
and go to (2); otherwise, terminate
```

Figure 1: The Procedure of BOA

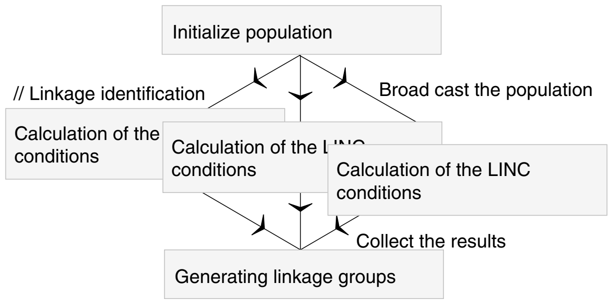
Initialize population
// Linkage identification
Broad cast the population
Calculation of the LINC conditions
Calculation of the LINC conditions
Calculation of the LINC conditions
Collect the results
Generating linkage groups

Figure 2: The procedure of pLINC

alent to those obtained by serial GAs in shorter time. The island model GAs run GAs with small population in some processors independently and exchange strings between processors every generation(s). Sometimes, they perform better than serial GAs, because building blocks in small populations increase faster and are combined through the exchange of strings.

These simple parallel GAs could not overcome the problem of simple GAs, i.e. building block disruptions. Building blocks (BBs) are highly fit sub-strings composed of interdependent variables and GAs must increase and combine them as whole blocks. If variables belonging to a BB are encoded loosely, one- (or multi-) point crossover might disrupt it. However, nevertheless the linked variables must be encoded tightly to combine BBs appropriately, such encodings are sometimes difficult.

2.2 Linkage and building blocks

It is considered that most of real-world problems are decomposable or quasi-decomposable into sub-problems. For such a (quasi-) decomposable function composed of some sub functions, BBs should be candidate solutions of the subfunctions. It is important to preserve and combine BBs appropriately. However, such important sub-strings are generally unknown. Therefore, there have been many efforts to detect variables tightly linked to form a BB automatically [8,9,12,15,20,21,23,25]. The relationship between such variables is called *linkage*. Competent GAs [4] construct probabilistic models and/or calculate fitness differences to identify linkage in order to solve problems quickly, reliably, and accurately. Linkage identification techniques which calculate fitness differences caused by perturbation of variables are called perturbation methods (PMs) and those which estimate the distribution of promising strings are called estimation of distribution algorithms (EDAs). Some cGAs are described in section 2.3.

2.3 Parallel competent genetic algorithms

To reduce the computational cost to identify linkage, parallel implementations of cGAs have been considered.

For example, PBOA, DBOA, parallel BOA [18,19] which are BOA [21] with the parallel Bayesian network construction, and pLINC [16] which is the parallel version of LINC [15] and pPADA [11] which is the parallel version of PADA have been proposed.

Bayesian optimization algorithm (BOA), which is one of EDAs or probabilistic model building GAs (PMBGAs) [10, 22], constructs a Bayesian network which fits promising strings and generates new strings from the network every generation. Figure 1 shows the algorithm of BOA. In the parallelization of BOA, not only fitness evaluations but also network constructions are parallelized. To construct feasible Bayesian networks (directed acyclic networks) in parallel, some tricks are required. PBOA and DBOA restrict edge directions in advance and parallel BOA performs backtracking after sub-networks are corrected from processors. Therefore, it is sometimes difficult to obtain models which are equivalent to the ones constructed by the original BOA.

Linkage Identification by Nonlinearity Check (LINC) [15], which is one of PMs, identifies linkage before the evolution stage of GAs. LINC calculates fitness differences by perturbations and judges whether any two variables (i, j) are independent or not by checking the non-linearity of the fitness differences. In concrete, if the following condition (1) is satisfied for all strings, LINC considers that (i, j) are independent; otherwise they are linked.

$$\begin{aligned} |\Delta f_{ij}(\boldsymbol{s}) - (\Delta f_i(\boldsymbol{s}) + \Delta f_j(\boldsymbol{s}))\,| &< \varepsilon, \qquad (1)\\ \Delta f_i(\boldsymbol{s}) &= f(\cdots \overline{s}_i \cdots\cdots) - f(\cdots s_i \cdots\cdots)\\ \Delta f_j(\boldsymbol{s}) &= f(\cdots\cdots \overline{s}_j \cdots) - f(\cdots\cdots s_j \cdots)\\ \Delta f_{ij}(\boldsymbol{s}) &= f(\cdots \overline{s}_i \cdot \overline{s}_j \cdots) - f(\cdots s_i \cdot s_j \cdots), \end{aligned}$$

where $\overline{s}_i = 1 - s_i$ and $\overline{s}_j = 1 - s_j$. Because LINC requires $O(l^2)$ fitness evaluations to check arbitrary pairs of l variables, the parallelized version of LINC (pLINC) has been proposed in [16]. In pLINC, several processors receive population from a master processor and perform the non-linearity checks to sent results to the master processor (Figure 2). Other linkage identification methods based on perturbations, such as Linkage Identification by non-Monotonicity Detection (LIMD), Linkage Identification with Epistasis Measures (LIEM) and Linkage Identification with Epistasis Measures considering Monotonicity (LIEM^2) [13–15] can be parallelized in a same way. Because the non-linearity checks in (1) can be performed independently in multiple processors, linkage models by pLINC are equivalent to these obtained by the original LINC. However, it is sometimes difficult to provide enough processors to catch up with the growth of problem size l.

3. PARALLELIZATION OF A GA USING D^5 AND CDC

Before considering the above mentioned parallel competent GAs, we show and parallelize another competent GA employing a linkage identification method called Dependency Detection for Distribution Derived from fitness Differences (D^5) [28] and a crossover method for complexly overlapping BBs called Context Dependent Crossover (CDC) [27].

```
Initialize population with $n$ strings

for $i = 1$ to $l$
   /* Calculate fitness differences */
   for each $\boldsymbol{s} \in P$
      $\boldsymbol{s}' = s_1 \cdots \overline{s}_i \cdots s_l$ /* Perturb $i$-th variable */
      $\Delta f_i(\boldsymbol{s}) = f(\boldsymbol{s}') - f(\boldsymbol{s})$
   end

   /* Cluster strings according to $\Delta f_i$ */
   classify $\boldsymbol{s} \in P$ into $C_1, C_2, \cdots$ according to $\Delta f_i(\boldsymbol{s})$

   /* Estimate clusters */
   for each $C_i$
      Find the set which gives the minimum entropy in
      $C_i$ and let it $V_i$.
   end
end
```

Figure 3: The Linkage Identification by D^5

```
Construct a graph $G = (N, E)$ where nodes correspond
to linkage sets and edges correspond to overlaps
between their nodes

For each pair of strings
  · Remove nodes where corresponding parental
    sub-strings are equal.
  · Remove edges where corresponding overlaps do not
    cause any BB disruption. Let the new graph $G'$.
  · Choose two nodes $n_1, n_2$ randomly from $G'$.
  · Partition $G'$ into $G_1 = (N_1, E_1)$ and $G_2 = (N_2, E_2)$
    s.t. $n_1 \in N_1$, $n_2 \in N_2$ and the number of cut edges
    are minimum
  · Let $\mathcal{V} = \bigcup_{V_j \in N_1} V_j$ and exchange variables in $\mathcal{V}$
end
```

Figure 4: The algorithm of Context Dependent Crossover

3.1 D^5

Like LINC, D^5 identifies linkage before the evolution stage of GAs. D^5 combines the mechanisms of EDAs and PMs to reduce fitness evaluations in existing PMs even for EDA-difficult problems. Figure 3 shows the algorithm of D^5. After initializing population, it investigates linkages for each variable i. The s_i in all strings $\boldsymbol{s}$ are perturbed to calculate fitness differences,

$$\Delta f_i(\boldsymbol{s}) = f(\cdots \overline{s}_i \cdots\cdots) - f(\cdots s_i \cdots\cdots).$$

Then, strings are classified according to $\Delta f_i(\boldsymbol{s})$. Each of the resulted clusters is estimated to obtain variables linked to i.

For decomposable functions $f(\boldsymbol{s}) = \sum_{V_j} f_j(\boldsymbol{s}_{\boldsymbol{v}_j})$ where f_j is a sub-function and V_j is a set of variables belonging to f_j, the difference $\Delta f_i(\boldsymbol{s})$ is defined only by variables linked to i:

$$\begin{aligned}\Delta f_i(\boldsymbol{s}) &= f(\cdots s_i \cdots) - f(\cdots \overline{s}_i \cdots)\\ &= \sum_{i \in V_j} f_j(\cdots s_i \cdots) - \sum_{i \in V_j} f_j(\cdots \overline{s}_i \cdots).\end{aligned}$$

After strings are clustered based on $\Delta f_i(\boldsymbol{s})$, the variables in the sets V_j such that $i \notin V_j$ take random values and their entropies become large. On the other hand, variables in V_j such that $i \in V_j$ take some particular sub-strings (sometimes a single sub-string) and their entropies should become small. Therefore, the set of variables which depend on i can be identified by finding the set of variables which minimizes the entropy.

After all linkage sets V_i $(i = 1, 2, \cdots, l)$ are obtained, they are collected to construct final linkage sets to be used by crossover operators. For problems with non-overlapping BBs, we can omit linkage identifications for variables which have been already found to link other variables. However, in the general problems, linkage identifications should be performed for all variables.

3.2 Context dependent crossover

If linkage sets have been obtained, we can consider a sub-string constructed of variables in a same linkage set as a BB candidate. For example, for a linkage set $\{1, 4, 5\}$, sub-strings $1{*}{*}11{*}{*}{*}\cdots$, $0{*}{*}11{*}{*}{*}\cdots$, $\cdots$, are BB candidates.

For overlapping BBs such as $1{*}{*}11{*}{*}{*}\cdot$ and ${*}{*}100{*}{*}{*}\cdots$, D^5-GA employs Context Dependent Crossover (CDC) [27] to combine them appropriately. CDC is the extension of the crossover method proposed by Yu et. al. [29]. To reduce BB disruptions even for complexly overlapping BBs, it considers the potential of the BB disruptions in detail by investigating values of parents in addition to the problem structure.

The algorithm of CDC is shown in Figure 4. The overlapping problem structure is represented by a graph G where nodes correspond to linkage sets (and BBs) and edges correspond to overlaps between linkage sets of their nodes. While the crossover by Yu et. al. uses only one graph throughout the optimization, CDC reconstruct the graph for each pair of strings. The nodes where corresponding BBs are identical in parents are removed and the edges where corresponding overlapping do not disrupt BBs are also removed. The reconstructions reduce BB disruptions and enrich crossover patterns even for problems with complexly overlapping BBs. For the reconstructed graph G, the min-cut that divides randomly selected two nodes is searched and BBs in one of the sub-graphs are exchanged to generate offsprings.

3.3 Parallelization of the D^5 and CDC

D^5 evaluates fitness to obtain fitness differences caused by perturbations. These are performed independently. Moreover, the clustering and estimation of clusters for each variable i are also independent. Therefore, it can be parallelized using a simple master-slave approach, in which each processor detects linkages for certain variables.

Figure 5 shows the algorithm of parallel D^5. First, a master processor sends population to other processors. Slave processors calculate fitness differences, cluster strings according to the differences, and estimate clusters to detect linkages for allocated variables. After all linkage identifications for the allocated variables are finished, the slave processors send their results to the master processor. The master processor constructs final linkage sets which are going to be used by crossover operators. For homogeneous systems, each processor detects linkages for l/n_p variables where n_p is the

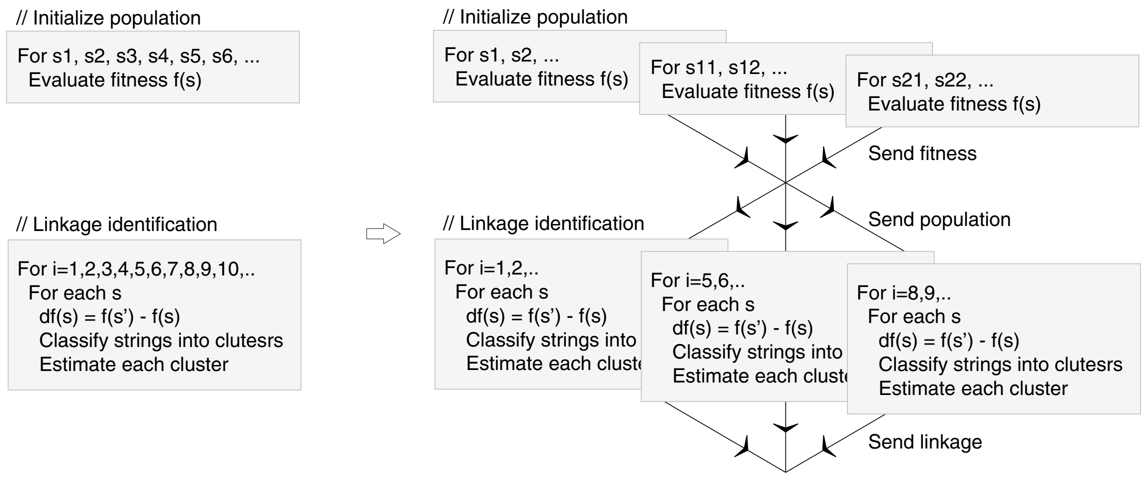

Figure 5: parallelization of D^5. The left one is the original algorithm and the right one is the parallelized D^5

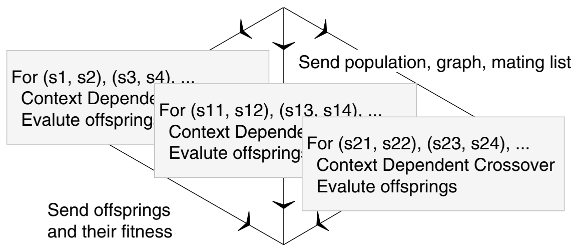
Send population, graph, mating list
For (s1, s2), (s3, s4), ...
Context Depende
Evalute offspring
For (s11, s12), (s13, s14), ...
Context Depende
Evalute offspring
For (s21, s22), (s23, s24), ...
Context Dependent Crossover
Evalute offsprings
Send offsprings
and their fitness

Figure 6: Parallelization of the evolution using CDC

number of processors. While the number of variables with respect to the ability of each processor will be assigned for heterogeneous systems, we consider only the homogeneous systems in this paper to make our investigation easy.

After linkage identification, population evolves in parallel (Figure 6). In each generation, a master processor sends population, mating lists and the original graph to other processors. The original graph represents overlapping relationships obtained by linkage sets. Each processor combines strings allocated by the mating lists using CDC – reconstructs the original graph and finds the min-cut — and evaluates offsprings. After all allocated crossovers are finished, the processors send offsprings and their fitnesses to the master processor. The master processor replaces parental strings by the offsprings using niching.

In following, we call this parallel D^5 or pD^5.

4. EXPERIMENTS AND DISCUSSIONS

In the following, we investigate the applicability and property of some of parallel cGAs —pD^5-GA , parallel BOA and pLINC-GA— experimentally. In our experiments, we focus on the total computational time and speed-up factor. The speed-up factor $S = T_s/T_p$ is calculated by the ratio of the time by serial processing T_s and time by parallel processing T_p. While most of the computational time will be spent on the fitness evaluations for computationally expensive functions, the other processes will effect the total computational time for computationally efficient functions. Therefore, we consider the computational time not the number of evaluations and we change not only problem size but also time to evaluate each fitness function.

First, we consider the computational times and speed-up factors of pD^5 and pLINC. The times to obtain optimal solution are recorded. While the LINC and D^5 identify linkages in different manners, they evolve strings in a similar way.

We employ the sum of 5-bit trap functions (3):

$$f(\boldsymbol{s}) = \sum_{j=1}^{m} \text{trap5}(\boldsymbol{s}_{\boldsymbol{v}_j}), \tag{2}$$

$$\boldsymbol{s}_{\boldsymbol{v}_1} = s_1 s_2 s_3 s_4 s_5, \boldsymbol{s}_{\boldsymbol{v}_2} = s_6 s_7 s_8 s_9 s_{10}, \cdots$$

$$\text{trap5}(\boldsymbol{s}_{\boldsymbol{v}_j}) = \begin{cases} \frac{4-u}{5}, & u = 0,1,2,3,4 \\ 1, & u = 5 \end{cases} \tag{3}$$

where $\boldsymbol{s}$ is string, m is the number of sub-functions (5-bit trap functions), f_j is j-th sub-function, $\boldsymbol{s}_{\boldsymbol{v}_j}$ is the sub-solution of the f_j, u is the number of ones in a sub-string $\boldsymbol{s}_{\boldsymbol{v}_j}$. In this experiment, there is no overlap between linkage sets. The string length is set to $l = 100, 200, 300$. The evaluation processes are suspended for $t_{sleep} = 0.00, 0.02$ and 0.04 seconds. The total computation time to obtain the optimal solution is recorded. Population sizes are varied in linkage identification phase and evolution phase and defined to obtain the optimal solution (Table 1). Crossover probabilities are $P_c = 1.0$ and Mutation rates are $P_m = 0.0$. Restricted tournament replacement (RTR) [7] is employed and the window size is $l/8$.

Table 1: Population size

	pD^5		pLINC	
l	linkage	evolution	linkage	evolution
100	400	300	12	300
200	450	380	20	380
300	450	500	25	500

All experiments are performed using 16 IBM x3455 computers (AMD Dual-Core processor 1.8GHz, Memory 1GB) connected via Gigabit Ethernet.

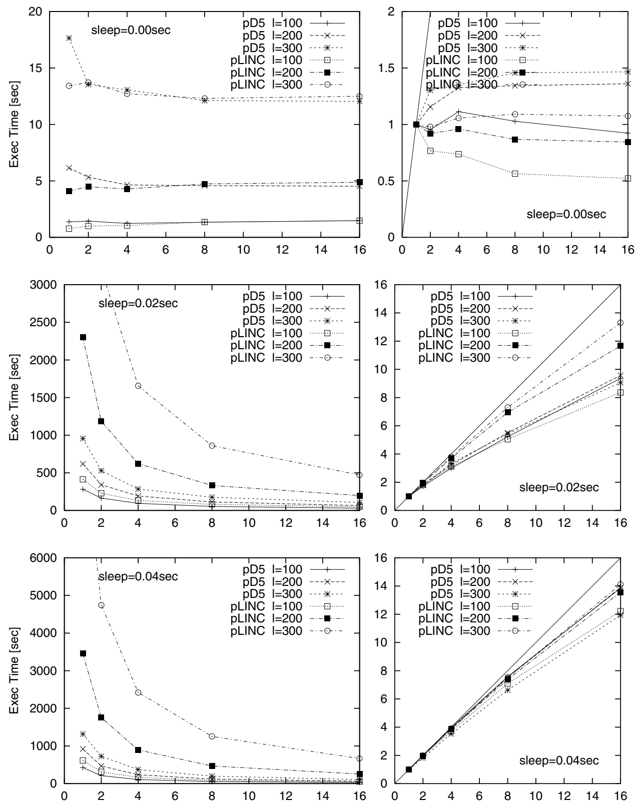

Figure 7: The total computational time (left) and speedup (right) of pD^5 and pLINC for trap function when $t_{sleep} = 0.00, 0.02, 0.04$ sec.

Figure 7 shows the computational time and speedup for each number of processors by pD^5 and pLINC for functions with $t_{sleep} = 0.00, 0.02, 0.04$ seconds.

When $t_{sleep} = 0.00$, pLINC finds the optimal solution faster than pD^5, because pD^5 performs some complex processes such as clustering and estimation to obtain linkage relationships. Since the fitness evaluations do not take time, the speed-up factors S of pD^5 and pLINC are small.

The left column of Figure 7 shows that when t_{sleep} and problem size become larger, pLINC takes longer execution time. This is because the original LINC requires $O(l^2)$ fitness evaluations where l is problem size. While pLINC delivers higher speed-up than pD^5 for $t_{sleep} = 0.02$, for $t_{sleep} = 0.04$, both of the pD^5 and pLINC deliver almost linear speed-up because the fitness evaluation time dominates in the total execution time.

Table 2: The estimated number of fitness function evaluations for pD^5 and pLINC for each problem size. Population size is shown in Table 1.

l	pD^5	pLINC
100	40,400	60,612
200	90,450	402,020
300	135,450	1,128,775

To understand the above phenomenon of the computational time of pD^5 and pLINC, we change t_{sleep} from 0 to 0.0001 seconds by little and little. Because the usleep() function which stops function evaluations cannot give exact suspend time for such small time, we count the number of function evaluations in each CPU and stop the product of the number and t_{sleep} all together. In this experiment, we consider only the linkage identification phase and do not perform the evolution phase. The sum of trap functions are employed again. The string length is $l = 100, 200, 300$. Population size is set as same as the previous experiment.

Figure 8 shows the computational times which pD^5 and pLINC require to detect linkage with respect to t_{sleep}. The top row is the case of the number of processors n_p is 1 and the bottom is for $n_p = 8$. Table 2 shows the estimated number of function evaluations for each linkage identification method. As shown in the table 2, pLINC spends more fitness evaluations for large problems. However, pD^5 requires more time to detect linkage than pLINC when function evaluation is completed instantly due to the complex process of its linkage identification. For example, for $l = 100$, pD^5 takes longer time when t_{sleep} is less than 7.0×10^{-5}, and pLINC takes longer time when t_{sleep} is more than 7.0×10^{-5}. For $l = 200$ or 300, pD^5 overtakes pLINC even when t_{sleep} is less than the case of $l = 100$. Moreover, these results are observed similarly in a single processor ($n_p = 1$) and in multiple processors ($n_p = 8$).

In the following, we consider the above situation theoretically. Because processors receive population and send linkage information in both of the pD^5 and pLINC, the process of each processor should be more critical than the communication. Therefore, we consider the process of each processor. Let t_f the time to evaluate one fitness function (the sum of t_{sleep} and real evaluation time), l problem size. Population size used by pLINC is n_l and that used by pD^5 is n_d. Because the number of fitness function evaluations in LINC is $\left(\frac{l(l-1)}{2} + l\right) n_l$ [15], pLINC takes

$$\left(\frac{l(l-1)}{2} + l\right) n_l \frac{t_f}{n_p}$$

seconds every processor. The other processes of pLINC take very short time, which can be approximated by 0.

The number of function evaluations in D^5 is ln_d [28] and the time for the evaluations is

$$ln_d \frac{t_f}{n_p}.$$

The other processes are clustering and estimation. They are performed l times. Let each clustering time t_c and estimation time t_e. Then, $\frac{(t_c+t_e)l}{n_p}$ is taken for them. Therefore, pD^5 takes

$$n_d \frac{t_f}{n_p} + \frac{(t_c + t_e)l}{n_p}$$

seconds every processor.

To simplify the analysis, we ignore the fact that t_e and t_c are not constant and function of l. Then, pD^5 finishes faster than pLINC when

$$\left(\frac{l(l-1)}{2} + l\right) n_l \frac{t_f}{n_p} > ln_d \frac{t_f}{n_p} + \frac{(t_c + t_e)l}{n_p}$$

$$t_f > \frac{(t_c + t_e)}{\left(\frac{l-1}{2} + 1\right) n_l - n_d}.$$

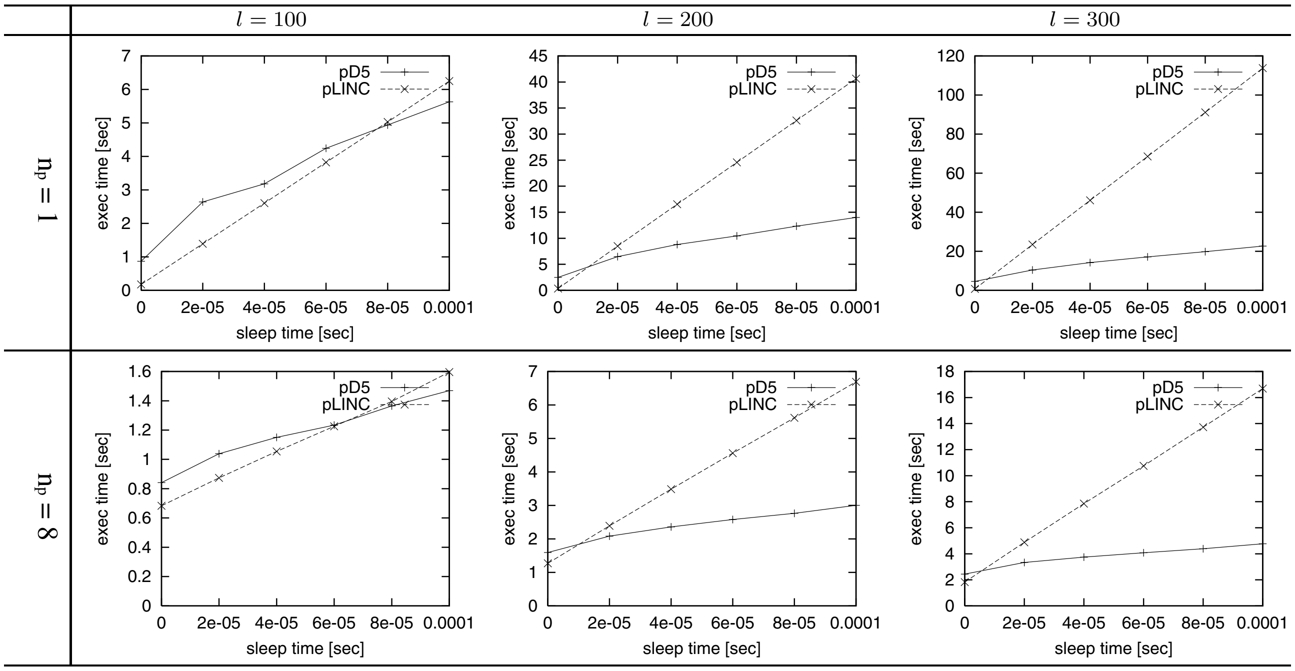

Figure 8: The total computational time for linkage identification per t_{sleep} of each fitness evaluation time

Because population sizes n_l, n_d increase $O(\log l)$ or less for l [15,26], when l becomes large, the denominator of the right side of the above equation increases and the threshold of t_f decreases.

Secondly, we perform experiments to compare pD^5-GA and parallel BOA [18] for functions composed of overlapping building blocks. In the following experiments, we focus on the performance in the evolution processes. While the parallel BOA detects linkage and evolves strings simultaneously, the pD^5-GA and pLINC-GA detect linkage in advance and then evolve strings. The evolving processes of the pD^5-GA and pLINC-GA are same and the linkage identification processes of them have been examined in the previous experiments, we compare the pD^5-GA and parallel BOA and omit the pLINC-GA here. Moreover, since the evolving processes are very easy for problems with non-overlapping building blocks, we change test problems from those with non-overlapping building blocks to those with overlapping building blocks. The parallel BOA, which showed better performance than DBOA in [18], constructs network in parallel and evaluates fitness in parallel. The parallel network construction is based on random addition of edges in parallel with a roll-back mechanism when it causes an infeasible network. The pD^5-GA is performed in a way shown in section 3.

We employ test functions composed of the sum of overlapping 5-bit trap functions used in [27]. Unlike the original 5-bit trap function defined in (2), the components of each trap function are chosen stochastically. The j-th trap function is composed of variables sampled as:

$$\begin{aligned} \boldsymbol{v}_j &= (N(3j, \sigma^2) \bmod l, N(3j, \sigma^2) \bmod l, \\ &\quad \cdots, N(3j, \sigma^2) \bmod l), \end{aligned} \tag{7}$$

where $N(\mu, \sigma^2)$ is the normal distribution with mean μ and variance σ^2. While for small σ^2 building blocks overlap only with its neighbouring ones, for large σ^2 they overlap randomly because their components are selected almost at random. For example, for small σ^2, trap functions are composed of

$$s_1 s_2 s_3 s_4 s_5,\ s_4 s_5 s_6 s_7 s_8,\ s_7 s_8 s_9 s_{10} s_{11}, \cdots$$

On the other hand, for large σ^2, they are

$$s_{10} s_{12} s_{15} s_{27} s_{38},\ s_{15} s_{17} s_{19} s_{22} s_{29},\ s_9 s_{15} s_{17} s_{21} s_{38}, \cdots$$

We consider $\sigma^2 = 1^2$ and $\sigma^2 = 5^2$ and $\mu = 3j$ ($j = 1, 2, \cdots$), $l = 60, 90$. Population sizes are defined to obtain the optimal solution for the pD^5-GA and parallel BOA and shown in Table 3. Fitness evaluation takes $t_{sleep} = 0.00, 0.04$ and 0.08 seconds.

Figure 9 shows the total computational times and speedup factors. For $t_{sleep} = 0.00$, parallel BOA takes longer computational time than pD^5 because the network construction performed every generation dominates in the computational time. Note that when shared memory is available, parallel BOA gives better speed-up as shown in [18] and elsewhere. However, because processors communicate several times every generation to construct network, parallel BOA cannot

Table 3: Population sizes for pD^5-GA and parallel BOA. (D^{5}l is for linkage identification and D^{5}e is for evolution)

σ^2	$l = 60$			$l = 90$		
	D^{5}l	D^{5}e	BOA	D^{5}l	D^{5}e	BOA
1^2	500	300	3000	600	400	5000∼6000
5^2	900	800	7000	1200	900	10000 ∼12000

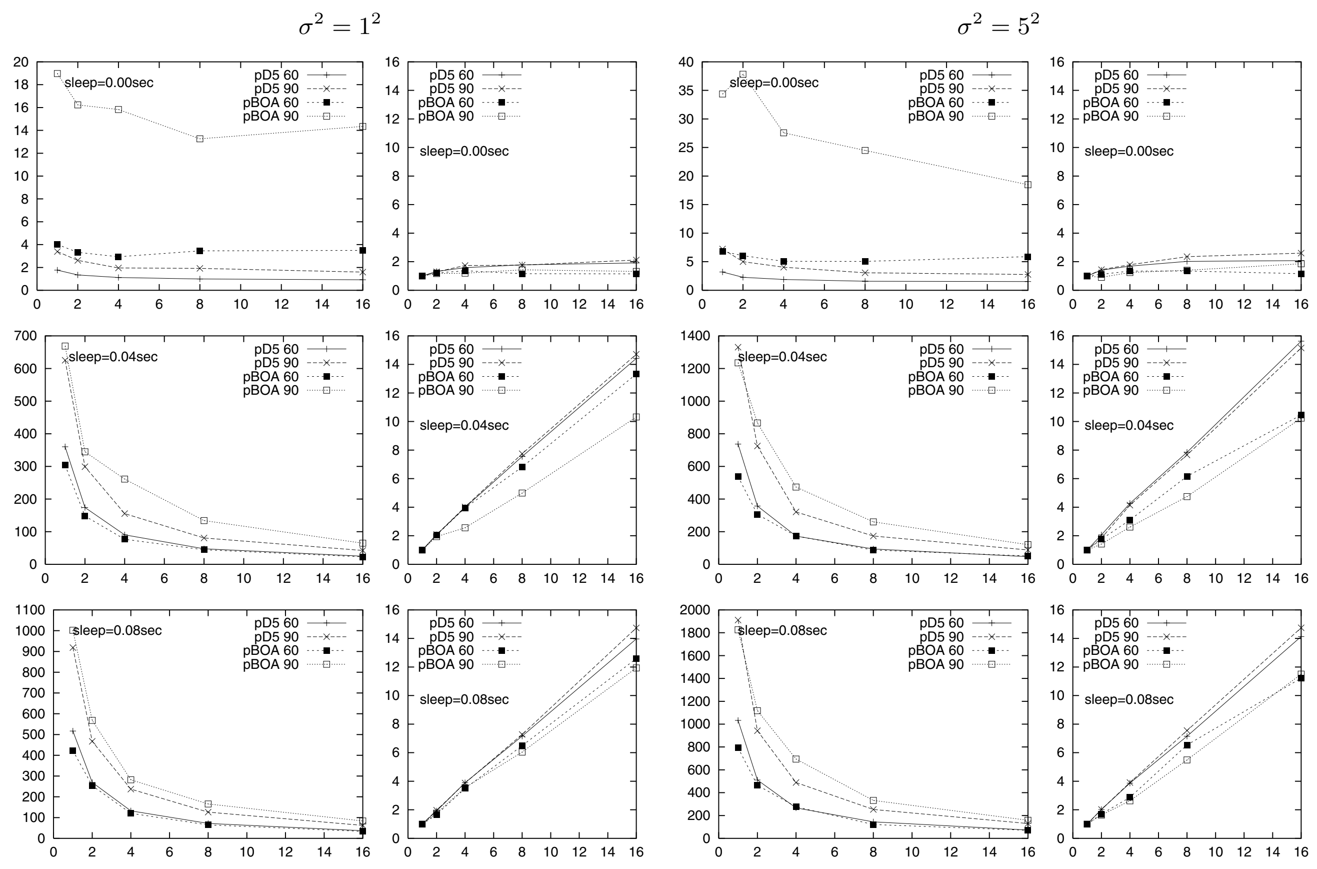

Figure 9: The total computational time and speedup for the functions with overlapping BBs ($\sigma^2 = 5^2$) by pD^5 and parallel BOA

result such good speed-up in a distributed memory architecture.

For $t_{sleep} = 0.04, 0.08$ and $\sigma^2 = 1$, the computational times of pD^5 and parallel BOA are almost equal, because the original pD^5 and BOA require $O(l \log l)$ and $O(l^{1.55})$ fitness evaluations respectively [24, 26] and there is no significant difference for relatively small problem size ($l = 60, 90$). When $\sigma^2 = 5^2$ and $t_{sleep} = 0.04, 0.08$, parallel BOA is faster in a single processor, which means the number of evaluations in the serial BOA is smaller. However, in multiple processors, the computational time of parallel BOA becomes almost equal to pD^5 for $l = 60$, and it becomes more than pD^5 for $l = 90$.

Looking at the speed-up factors, pD^5 results more speed-up for $t_{sleep} = 0.04$ and 0.08 because the roll-back mechanism in parallel BOA cannot make equivalent Baysian network to the one from original BOA. Therefore, the number of generations and strings in parallel BOA increases when the number of processors increases. For $t_{sleep} = 0.08$, the difference of the speed-up is small because fitness evaluations take most part of the computational time in both algorithms.

5. CONCLUSION

We have empirically investigated parallel competent GAs including pD^5-GA which has been parallelized in this paper. D^5-GA can be parallized based on a master-slave approach because the linkage identification by D^5 can be performed independently for each variable. For functions which can be evaluated in no time, pD^5 takes longer computational time than pLINC due to the complexity of its mechanism. On the other hand, pD^5 detects linkage faster than pLINC for computationally expensive functions because pLINC requires many function evaluations. These results are observed similarly in a single processor and in multiple processors. At a certain amount of time to evaluate fitness, pLINC and pD^5 finish at a same time. The threshold is considered experimentally and theoretically, and it is shown that pD^5 catches up with pLINC in a shorter time for a larger problem. Moreover, while the speedup factor of parallel BOA is limited due to the quality of models constructed in parallel, that of pD^5-GA rises with the growing fitness evaluation time.

Further research should investigate the performance of parallel cGAs for other systems such as heterogeneous systems, systems with shared memory. In addition, for real-world applications, parallel cGAs for real-valued strings should be investigated.

6. REFERENCES

[1] E. Alba and M. Tomassini. Parallel and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002.

[2] A. D. Bethke. Comparison of genetic algorithms and gradient-based optimizers on parallel processors: Efficiency of use of processing capacity. Technical report, University of Michigan, 1976.

[3] E. Cantú-Paz. *Designing Efficient and Accurate Parallel Genetic Algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, 1999.

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

[5] J. J. Grefenstette. Parallel adaptive algorithms for function optimization. Technical Report CS-81-19, Vanderbilt University, 1981.

[6] P. B. Grosso. *Computer simulations of genetic adaptation: parallel subcomponent interaction in a multilocus model.* PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1985.

[7] G. R. Harik. Finding multimodal solutions using restricted tournament selection. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 24–31, San Francisco, CA, 1995. Morgan Kaufmann.

[8] R. B. Heckendorn and A. H. Wright. Efficient linkage discovery by limited probing. In *Proceedings of the Genetic and Evolutionary Computation Conference Part 1, LNCS 2723*, pp. 1003–1014, 2003.

[9] H. Kargupta and B. Park. Gene expression and fast construction of distributed evolutionary representation. *Evolutionary Computation*, 9(1):43–69, 2001.

[10] P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, 2001.

[11] J. Madera, E. Alba, and G. Luque. Performance evaluation of the parallel polytree approximation distribution algorithm on three network technologies. *Inteligencia Artificial, Revista Iberoamericana de IA*, 11(35):67–76, 2007.

[12] H. Mühlenbein and T. Mahnig. The factorized distribution algorithm for additively decomposable functions. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 752–759, 1999.

[13] M. Munetomo. Linkage identification based on epistasis measures to realize efficient genetic algorithms. In *Proceedings of the Congress on Evolutionary Computation*, pp. 445–452, 2002.

[14] M. Munetomo. Linkage identification with epistasis measure considering monotonicity conditions. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*, 2002.

[15] M. Munetomo and D. E. Goldberg. Identifying linkage groups by nonlinearity/non-monotonicity detection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 433–440. Morgan Kaufmann Publishers, 7 1999.

[16] M. Munetomo, N. Murao, and K. Akama. A parallel genetic algorithm based on linkage identification. In *Proceedings of the Genetic and Evolutionary Computation, LNCS 2723*, pp. 1222–1233, 2003.

[17] M. Munetomo, N. Murao, and K. Akama. Empirical investigations on parallelized linkage identification. In *Proceedings of the Parallel Problem Solving from Nature, LNCS 3242*, pp. 322–311. Springer, 2004.

[18] M. Munetomo, N. Murao, and K. Akama. Empirical studies on parallel network construction of bayesian optimization algorithms. In *Proceedings of Congress on Evolutionary Computation, Vol.2*, pp. 1524–1531. IEEE, 2005.

[19] J. Očenášek. *Parallel estimation of distribution algorithms.* Doctroral dessertation, Faculty of Information Technology, Brno University of Technology, 2002.

[20] M. Pelikan and D. E. Goldberg. Hierarchical problem solving and the Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO 2000*, pp. 267–274. Morgan Kaufmann Publishers, 2000.

[21] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO1999*, pp. 525–532. Morgan Kaufmann Publishers, 1999.

[22] M. Pelikan, D. E. Goldberg, and F. G. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20, 2002.

[23] M. Pelikan and H. Mühlenbein. Marginal distribution in evolutionary algorithms. In *Proceedings of the International Conference on Genetic Algorithms Mendel '98*, pp. 90–95, Brno, Czech Republic, 1998.

[24] M. Pelikan, K. Sastry, and D. E. Goldberg. Scalability of the Bayesian optimization algorithm. *International Journal of Approximate Reasoning*, 31(3):221–258, 2002.

[25] M. Tsuji, M. Munetomo, and K. Akama. Modeling dependencies of loci with string classification according to fitness differences. In *Proceedings of the Genetic and Evolutionary Computation Conference Part 2, LNCS 3103*, pp. 246–257. Springer-Verlag, 2004.

[26] M. Tsuji, M. Munetomo, and K. Akama. Population sizing of dependency detection by fitness difference classification. In *Foundations of Genetic Algorithms, 8th International Workshop, LNCS 3469*, pp. 282–299, 2005.

[27] M. Tsuji, M. Munetomo, and K. Akama. A crossover for complex building blocks overlapping. In *Proceedings of the Genetic and Evolutionary Computation*, pp. 1337–1344, 2006.

[28] M. Tsuji, M. Munetomo, and K. Akama. Linkage identification by fitness difference clustering. *Evolutionary Computation*, 14(4):383–409, 2006.

[29] T.-L. Yu, K. Sastry, and D. E. Goldberg. Linkage learning, overlapping building blocks, and systematic strategy for scalable recombination. In *Proceedings of the Genetic and evolutionary computation conference*, pp. 1217–1224, 6 2005.