# Mega Process Genetic Algorithm Using Grid MP

Yoshiko Hanada[1], Tomoyuki Hiroyasu[2], Mitsunori Miki[2], and Yuko Okamoto[3]

[1] Graduate School, Department of Knowledge Engineering and Computer Sciences, Doshisha University, 610-0321 Kyoto, Japan
hanada@mikilab.doshisha.ac.jp
[2] Department of Knowledge Engineering and Computer Sciences, Doshisha University, 610-0321 Kyoto, Japan
{tomo@is, mmiki@mail}.doshisha.ac.jp
[3] Department of Theoretical Studies Institute for Molecular Science, Okazaki, Aichi 444-8585, Japan
okamotoy@ims.ac.jp

**Abstract.** In this study, a new Genetic Algorithm (GA) using the Tabu · Local Search mechanism is proposed. The GA described in this paper is considered a Mega Process GA, which has an effective mechanism to use massive processors, i.e., Mega Processors, in large-scale computing systems. Our proposed method has a GA-specific database that possesses information of searched space and performs a local search for the space that is not searched. Such mechanisms enable us to comprehend the quantitative rate of a searched region during the search. Using this information, the searched space can be expanded linearly as the number of computing resources increases and the exhaustive search is guaranteed under infinite computations. The proposed GA was applied to numerical test functions and the energy minimization problems of protein tertiary structures. The latter problem was performed under a heterogeneous distributed computing environment, which was built up with Grid MP produced by United Devices Inc.

## 1 Introduction

Genetic Algorithms (GAs) are among the most effective approximation algorithms for optimization problems[1]. Various mechanisms for improving GAs have been discussed. Minimal Generation Gap (MGG)[2] was proposed as a generation alternation model. Methods using Linkage Identification[3, 4], Real-coded GA[5], Probabilistic Model-Building GAs[6, 7], and Distributed GA[8] are other GAs that have strong search capabilities. The restart mechanisms have also been applied to enhance the performance of GAs[9, 10, 11, 12]. However, application of a GA to solve optimization problems has the drawback that GAs incur large computing costs. One solution to this problem is to perform GAs in parallel. In recent decades,

A. Konagaya and K. Satou (Eds.): LSGRID 2004, LNAI 3370, pp. 152–170, 2005.

due to the remarkable improvements in computing capabilities, some parts of the drawback regarding computing costs of GAs are not really important. Furthermore, parallel processing is used to yield increases in performance of GAs. Recently, because of the emergence of super PC clusters and Grid computation environments, such as PC Grid comprised of desktop machines for home use or offices, the number of computational calculation resources is increasing. Thus, large computing projects in fields relating to evolutionary computing have become feasible. GAs are well suited to parallel processing environments due to the ability to search with multiple points, and consequently GAs have found application in large-scale computing[13, 14, 15, 16]. However, adapted methods are mostly simple parallelization of conventional GAs, which have been proposed for limited computing resources and an effective mechanism to make use of huge computing resources have yet to be designed. The easiest way that is commonly used for conventional GAs to use many resources is to increase the population size. However, enlarging the population also increases the diversity of the solutions, i.e., the convergence speed becomes slow. Subsequently, when GAs use a large amount of computer resources, the optimum solution is not derived quickly. At the same time, for conventional GAs, there is no guarantee of an exhaustive search in all search space although infinite computations are performed. As a result, although simple parallelization is applied to these methods, there is no assurance of improvement of their performance in accordance with the increase in available computing resources.

In this study, a new GA using the Tabu · Local Search mechanism for large-scale computer systems is proposed. We call such a GA using huge computing resources a Mega Process GA and our approach is to develop an effective mechanism to use massive processors, i.e., Mega Processors, in large-scale computing systems, such as super PC clusters and Grid computation environments. The proposed method uses a database that possesses information of space that has been searched already. At the same time, the proposed GA performs the local search for the space that is not searched to expand the searched space. These mechanisms enable us to comprehend the quantitative rate of the searched region during the search. Moreover, using this information, the searched space increases linearly as the number of computing resources increases and an exhaustive search is guaranteed under infinite computations. In addition, we examined the performance of the proposed method in a distributed computing environment, which is built up using the commercially available PC Grid middleware Grid MP that is currently used to develop several large-scale distributed computing projects produced by United Devices Inc[1].

## 2 Tabu · Local Search Mechanism for Mega Process GA

### 2.1 Database Structure

In this study, we introduce a GA-specific database that possesses information of the region that has already been searched. We used binary-coded individuals. In

[1] United Devices : http://www.ud.com

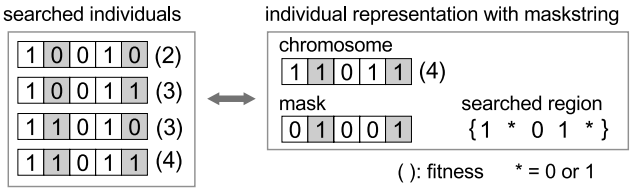



**Fig. 1.** An Example of an Individual stored in a Database

| | Chromosome | Mask | Searched Region | Fitness | Hamming-Index |
|---|---|---|---|---|---|
| Individual $x_1$ | 1 0 1 1 1 | 1 0 0 0 1 | * 0 1 1 * | 4 | 2 |
| Individual $x_2$ | 1 0 0 1 1 | 0 0 0 1 1 | 1 0 0 * * | 3 | 2 |
| Individual $x_3$ | 0 1 1 1 1 | 0 1 1 0 1 | 0 * * 1 * | 4 | 3 |
| Individual $x_4$ | 1 1 0 0 1 | 0 0 0 0 0 | 1 1 0 0 1 | 3 | 0 |

* = 0 or 1

**Fig. 2.** The Aspect of the Database of the proposed method

addition to chromosomes, individuals stored in the database possess bitstrings, which we call maskstrings. Maskstrings are also bitstrings, lengths of which are the same as those of chromosomes. A locus of a chromosome, which stands at '1' in a maskstring, represents that it has already searched all assignable genes to it. Fig. 1 shows that a set of searched individuals is compressed to one individual using a maskstring.

Our proposed database stores these individuals that hold maskstrings. Fig. 2 shows an instance of the database. The individual $x_3$, stored in the database shown in Fig. 2 implies that "0 1 1 1 1" is the best solution under searching the set of individuals $X_3 = \{0\ *\ *\ 1\ *\}(* = 0 \text{ or } 1)$. We call the number of '1' in a maskstring the hamming-index, e.g. the hamming-index of the individual $x_3$ is 3.

When a database stores all the individual information, it takes a large time to check an individual that has been already searched due to its vast amounts of data. Our proposed notation of searched individuals is a highly compressed method using maskstrings and large searched regions are represented by several individuals. Moreover, checking individuals stored on the database is not time-consuming.

This notation of individuals enables us to provide a quantitative rate of a searched region during a search. In an individual $x$ we denote its chromosome length by $L$, a gene of the locus $l$ of a chromosome by $c_{xl}$, and a value of the locus $l$ of a maskstring by $m_{xl}$. Quantitative sizes of searched regions are obtained as follows.

**Case of one Individual.** $|X|$ denotes a number of elements involved in a set $X$. The size of the searched region is indicated by an individual of which the hamming-index standing at $h$ is $2^h$, e.g., $|X_3|$, which is the size of the searched region indicated by the individual $x_3$ stored in the database shown in Fig. 2, as $2^3$.

**Case of *N* Individuals.** Given $N$ individuals $x_i (1 \leq i \leq N)$ and their search regions $X_i$, the total searched region indicated by them is the union of sets $|X_1 \cup X_2 \cup \cdots \cup X_N|$, i.e., $|\bigcup_{i \in I} X_i|$, where the set which consists of the suffix $i$ is denoted $I = \{1, 2, \cdots, N\}$. In most cases, it cannot be derived readily as a function of the number of elements in a union of sets. On the other hand, that of an intersection of sets is a closed form. The size of the searched region is such a case.

As a result, $|\bigcup_{i \in I} X_i|$ is derived from the sets of $|\bigcap_{j \in J} X_j|$, where $J$ is a subset of $I$, which is shown in equation (1).[2]

$$\left|\bigcup_{i \in I} X_i\right| = \sum_{J \subseteq I, J \neq \phi} (-1)^{|J|-1} \left|\bigcap_{j \in J} X_j\right| \tag{1}$$

The distance between individuals $x_i$ and $x_j$ including their maskstrings, which is represented as $d(x_i, x_j)$, is defined in (2).

$$\begin{aligned} d(x_i, x_j) &= \sum_{l=1}^{L} B|c_{x_i l} - c_{x_j l}| \\ B &= \begin{cases} 1, \text{if} \quad m_{x_i l} = m_{x_j l} = 0 \\ 0, \text{otherwise} \end{cases} \end{aligned} \tag{2}$$

$|\bigcap_{i \in I} X_I|$ is a closed form and derived below as (3), where $[R]{=}z$ given real number $R$ and integer $z$.

$$\begin{aligned} |X_1 \cap X_2 \cap \cdots \cap X_N| &= \begin{cases} 2^M, \text{ if } K = 0 \\ 0, \quad \text{otherwise} \end{cases} \\ K &= \sum_{i=1}^{N-1} \sum_{j=(i+1)}^{N} d(x_i, x_j) \\ M &= \sum_{l=1}^{L} \left[ \frac{1}{N} \sum_{i=1}^{N} m_{x_i l} \right] \end{aligned} \tag{3}$$

## 2.2 Concept of the Proposed Method

The proposed method consists of a GA and a local search. The flow of our proposed method is shown in Fig. 3. To obtain optima earlier, our method of searches

[2] For instance, the number of elements of the union of three sets, $X_1$, $X_2$, and $X_3$, is obtained as follows:
$|X_1 \cup X_2 \cup X_3| = |X_1| + |X_2| + |X_3| - |X_1 \cap X_2| - |X_2 \cap X_3| - |X_3 \cap X_1| + |X_1 \cap X_2 \cap X_3|$

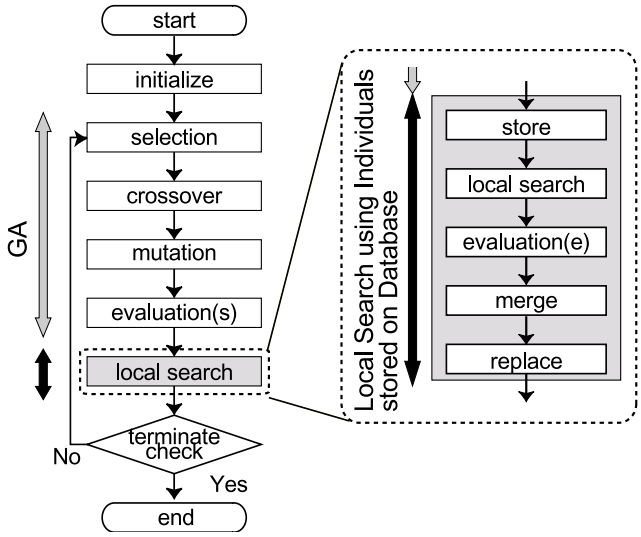



**Fig. 3.** The Flow of the Proposed Method

used mainly schemes of GA. Any method of operators, such as a crossover and mutation, or a generation alternation model can be applied. To use idle computing resources of enormous computing environments effectively, a local search is applied. Our proposed method is outlined as follows.

**Step 1.** Generate $N_{pop}$ individuals randomly, where $N_{pop}$ is the population size. In addition, no individuals are stored in the database.

**Step 2.** Apply operators, such as crossovers, mutations, and selections to individuals in a GA population.

**Step 3.** Store the individual $y_{best}$, which is the best individual of a GA population, in the database and set its maskstring to the bitstring of which the values of all loci are '0'. However, $y_{best}$ is not stored when it is included in the searched regions, which are indicated by individuals that have already been stored in the database.

**Step 4.** /Local search/ Expand a searched region indicated by a certain individual stored in the database. When a better individual is found, use it to replace the worst individual of the population of GA. The details of the local search are described in the next section 2.3.

**Step 5.** Go back to step 2 until some termination conditions, e.g., computing cost reaches a limited amount or the exhaustive search is done, are satisfied.

At step 3, when there are $N_{DB}$ individuals in the database, replace the individual $x_{worst}$ that has the poorest fitness in the database with $y_{best}$ if the fitness

of $y_{best}$ is bigger than that of $x_{worst}$, otherwise $y_{best}$ is not stored, where $N_{DB}$ is the parameter of database capacity.

### 2.3 Local Search

We chose a certain individual from the database to apply the local search in every generation. This individual satisfies the condition that the hamming-index is the minimum value and its fitness is the maximum value. In our proposed local search, one of the loci, the values of which stand at '0' of the maskstring of the selected individual, is changed to a value of '1'. The locus that holds the largest variance of genes among all loci is selected. This operation is performed because it is suitable to keep essentially the same hamming-index among individuals stored in the database for the process of merging individuals, which is described later. Moreover, it is desirable to retain genes of loci, which have lower variances as they can be considered part of a better solution.

Given $N$ individuals $x_i (1 \leq i \leq N)$ stored in the database, the process of the local search is outlined as follows. Fig. 4 shows an example of the local search.

**Step 1.** Find the results of $a_l$ using equation (4), which is the absolute value of the difference between the average value of genes and 0.5 at each locus.

$$a_l = \left| \frac{1}{N} \sum_{i=1}^{N} (c_{x_i l}) - 0.5 \right| \quad (1 \leq l \leq L) \tag{4}$$

**Step 2.** Select the individual $x$, which has the minimum hamming-index and maximum fitness, from $N$ individuals stored in the database.

**Step 3.** Select the locus $l^*$, which indicates $m_{xl^*} = 0$. At the same time, $a_{l^*}$ is the minimum value, from $a_l (1 \leq l \leq L)$.

**Step 4.** Prepare the individual $x'$, which has the same maskstring as that of $x$. In its chromosome, each gene is exactly the same as that of $x$ at loci standing at '1' except $l^*$ in its maskstring. $x'$ is the best individual under search $X'$, which should be searched to expand the search region.

**Step 5.** Update $m_{xl^*}$ to '1' and $h_x$ to $h_x$+1. Furthermore, let $x$ be $x'$ and replace the worst individual of the population of GA by $x$ if $x < x'$. The exhaustive search is finished when $h_x$ approaches $L$.

At step 4, described above, $X'$ is comprised of individuals produced by flipping the gene at $l^*$ in the chromosome of each individual included in $X$. This requires searching of $2^{h_x}$ individuals where the hamming-index of $x$ is $h_x$. Nevertheless, parallelization can be applied to easily search $X'$. Moreover, this mechanism uses computing resources effectively as the searched space increases linearly with increasing computing resources and an exhaustive search is guaranteed under infinite computations.

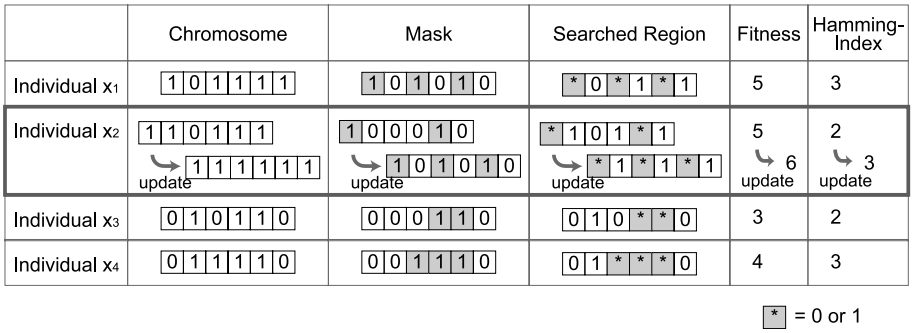



**Fig. 4.** An Example of Local Search in the one max problem: The individual of which the search region is expanded is individual $x_2$. Expanding the region $X_2$={* 1 0 1 * 1 } to the region {* 1 * 1 * 1 } requires searching the region $X_2'$={* 1 1 1 * 1 }. The searched region of $x_2$, i.e., $X_2$, becomes {* 1 * 1 * 1 }, after applying this expansion

### 2.4 Merge Operation in Database

Following the local search, a merge of individuals stored in the database is executed in every generation to avoid overlapping searches in the process of the local search. Individuals can be merged when the following conditions are satisfied:

**Condition 1.** $x_a$ and $x_b$ are given individuals. When they satisfy the following conditions, they are in condition 1. They have the same maskstrings, $d(x_a, x_b) = 1$, and locus is $l^*$ , which satisfies $m_{x_a l^*} = m_{x_b l^*} = 0$ and $c_{x_a l^*} \neq c_{x_b l^*}$. In this case, select one that has better fitness from $x_a$ and $x_b$, and let its value of $l^*$ in its maskstring be '1'. At the same time, the other is deleted from the database. For example, in Fig. 4, the individual $x_1$ and the updated individual $x_2$ can be merged with the condition 1 then let $m_{x_2 2}$ be '1' and $h_{x_2}$ be '4'. $x_1$ is then deleted.

**Condition 2.** $x_a$ and $x_b$ are given individuals. When these individuals satisfy the following conditions, they are in condition 2: $d(x_a, x_b) = 0$ and no locus exists that satisfies $m_{x_a l} = 1, m_{x_b l} = 0 (1 \leq l \leq L)$. In this case, $x_a$ is deleted from the database because of $X_a \subset X_b$. For example, in Fig. 4, the individual $x_3$ and the individual $x_4$ can be merged meeting condition 2. Therefore, $x_3$ is deleted.

**Condition 3.** $x_a$ and $x_b$ are given individuals. When $X_a \cap X_b \neq \phi$ and $|X_a| \geq |X_b|$, they are in condition 3 shown Fig. 5. In this case, $X_a$ is expanded until it can include $X_b$, and then they are merged using condition 2. $X_a'$ indicates the region that is required to expand until it can include $X_b$. $X_a' \cap \neg X_b$ must be searched to merge using condition 2. We introduce parameter $A$, which indicates the ratio of $|X_a \cap X_b|$ of $|X_a' \cap \neg X_b|$, i.e., $|X_a \cap X_b| / |X_a' \cap \neg X_b|$, because we obtain a better solution under the available limited computing resources. This merge is

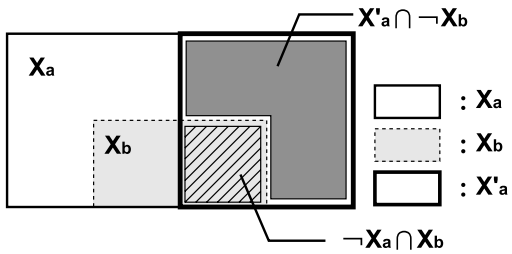



**Fig. 5.** Merge using Condition 3

not applied when $A$ is larger than a certain value. To decide an appropriate $A$, the size of the search space and the number of available computing resources or cost must be considered. In the present study, we set $A$ to 8.

## 3 Implementation of Mega Process GA on Distributed Computing Environments

### 3.1 Distributed Processing at Local Search

Parallelization is applicable to GA by using a master-slave model at evaluations or crossovers. However, when the population becomes larger, the diversity of the solutions also increases. As a consequence of this, huge computing resources cannot be used effectively. Our proposed local search is applied to use massive processors, i.e., Mega Processors.

At the local search phase in our proposed method, application of a local search to individual $x$ corresponds to searching the individual $x'$ of which the maskstring is the same as that of $x$. It also satisfies $d(x, x') = 1$, i.e., the set of individuals that have arbitrary values at loci of the chromosome standing at '1' in the maskstring of $x'$ and the same values as those of $x$ at other loci should be searched. In the example shown in Fig. 4, for application of the local search to $X_2$={* 1 $\underline{0}$ 1 * 1 } at locus 3 it is necessary to search $X_2'$={* 1 $\underline{1}$ 1 * 1 }(* = 0 or 1).

By using the opposite operation against condition 1 of the merger, parallelization can be applied easily to evaluation of a set of individuals represented with a maskstring. Fig. 6 shows an example in which a set of individuals is split to some segmental sets of individuals.

To execute the local search in distributed computing environments, these parts of the set are allotted to computation nodes. Each node evaluates assigned individuals independently, which requires no communication among nodes.

### 3.2 Implementation on Grid MP

We examine the performance of the proposed method in a distributed computing environment, built using the commercially available middleware Grid MP from

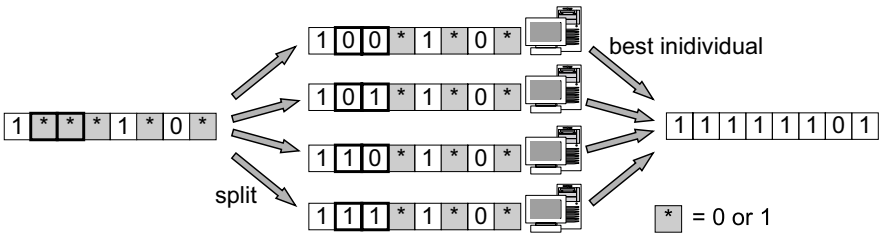



**Fig. 6.** Example of Splitting Individuals

United Devices Inc.[17]. Grid MP is one of the toolkits of Grid computing and is currently used to power several large-scale distributed computing projects. In the Grid MP platform, the underutilized resources of many computers are aggregated and used as a virtual computer system. The distributed computing environment constructed by Grid MP consists of an MP Server and Devices. The MP Server is a server that carries out user or Device authentication, monitoring and scheduling of jobs, dispatching jobs to Devices, etc. Devices are computation nodes that execute the jobs assigned by the MP Server.

In the Grid MP platform, application developers prepare their Program Module executables, which are components of applications and consist of precompiled executables and MDFs (Module Definition Files), and upload them to the File Service of the MP Server. Data Packages, which are sets of reference data during executions and PMFs (Package Manifest Files), have to also be registered as a Data Set. MDFs indicate names of the executable, arguments, and the file in which the results are written. PMFs show the compression format and the file name of the data. A Job object comprising of several Workunits is created once a user submits a Job. A Workunit is the minimum unit of a Job that one Device has to execute and defines the Program Module executable and the Data to be used. A Device during the polling state is assigned one Workunit. Fig. 7 shows the standard Job execution form of Grid MP.

To implement our proposed local search on distributed computing environments as shown in Fig. 6, the Program Module, which reads a data file in which individuals to be searched are written and searches these individuals, must be prepared.

### 3.3 Overhead of Grid MP

We performed our proposed method in the heterogeneous distributed computing environment composed of machines belonging to RIKEN Genomic Sciences Center (GSC)[3] and Intelligent Design Systems Laboratory (ISDL) of Doshisha University[4]. We used Grid MP platform version 4.0-3106. The specification of machines used for the experiments is shown in Table 1. The User Machine in Table 1 indicates the machine of a user who submits a job to the MP Server.

[3] RIKEN GSC : http://big.gsc.riken.jp/
[4] ISDL of Doshisha Univ. : http://mikilab.doshisha.ac.jp/

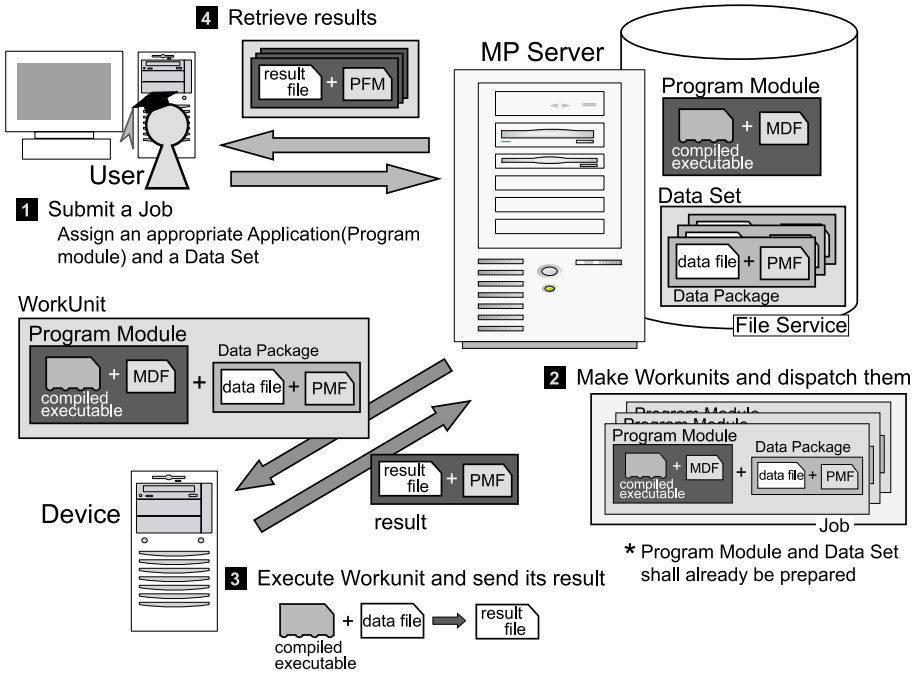


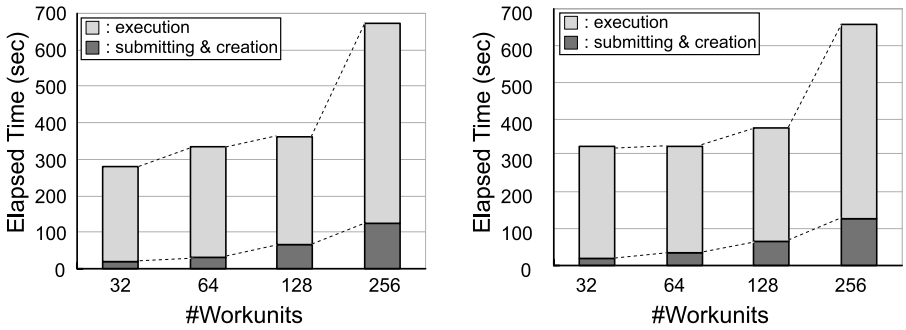

**Fig. 7.** Aspect of Performance of Job Execution form of Grid MP

**Table 1.** Specification of Machines Used for the Experiment

| | Affiliation | Machine name | #Nodes | Processor | Memory |
|---|---|---|---|---|---|
| User Machine | - - | harrier | 1 | mobile Pentium 1.8GHz | 1GB |
| MP Server | Doshisha ISDL | naiad | 1 | Xeon 2.8GHz x 2 | 2GB |
| Devices | | forte01-15 | 15 | PentiumIII 600MHz | 128MB |
| | | libra/tiger | 2 | Xeon 2.8GHz x 2 | 1GB |
| | RIKEN GSC | le01-23 | 23 | Celeron 1.3GHz | 896MB |

There is overhead due to the distinctive characteristics of the distributed computing environment, such as communication environment and latency of the middleware architecture dispatch mechanism. The latency is caused by some processes on the MP Server and the Devices, e.g., creation of a Job object and a schedule of Workunits and downloads of Workunits from the File Service of the MP Server.

We examined elapsed times of execution of the Job, the Workunit of which was only assigned and required no calculation in Devices. The number of Workunits included in the Job was set to 32, 64, 128, and 256. Polling interval was set to 30 seconds and 1 minute. 30 seconds is the minimum feasible interval. Fig. 8 shows the average value of Job creation time and its execution time. The results shown are from 10 trials.




(a) Polling Interval: 30sec (b) Polling Interval: 1min

**Fig. 8.** Elapsed Times of Empty Job

A Workunit is allotted to a Device, which has sent idle state of CPU to the MP Server several times. As a consequence of this, execution time is expected to depend on polling interval. However Fig. 8 illustrates that this interval has no effect much on elapsed times. It indicates that the execution time is taken up by preliminary processes such as preparing Workunits for their assignments rather than by Devices queuing.

From Fig. 8 the period time of Job creation increases exponentially as increasing in Workunits, in contrast execution time will not increase. This is because once the Device completes its assigned Workunit and sends result to the MP Server, it obtains a Workunit again if incomplete Workunits stay still in the File Service. Therefore the latency of dispatching is hidden seemingly.

## 4 Numerical Experiments

To discuss the effectiveness of our proposed method in both infinite and limited computation, it was applied to the one max problem and 3-deceptive problem[18]. The former is the most primitive benchmark problem of bitstrings, and its fitness is a summation of the number of '1' included in a chromosome. The latter is one of trap functions described as equation (5):

$$F_{3-deceptive} = \sum_{i=1}^{N} f_i \tag{5}$$

$$f_i = \begin{cases} 0.9, & u_i = 0 \\ 0.8, & u_i = 1 \\ 0.7, & u_i = 2 \\ 1.0, & u_i = 3 \end{cases}$$

The effectiveness of our method is discussed by solving these problems with limited computation. The following equations (6), (7), (8), and (9) are the continuous test functions:

$$F_{Rastrigin} = \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right) \tag{6}$$
$$x_i \in [-5.12, 5.12)$$

$$F_{Schwefel} = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{|x_i|}\right) \tag{7}$$
$$x_i \in [-5.12, 5.12)$$

$$F_{Ridge} = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2 \tag{8}$$
$$x_i \in [-64, 64)$$

$$F_{Griewank} = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \left(\cos\left(\frac{x_i}{\sqrt{i}}\right)\right) \tag{9}$$
$$x_i \in [-512, 512)$$

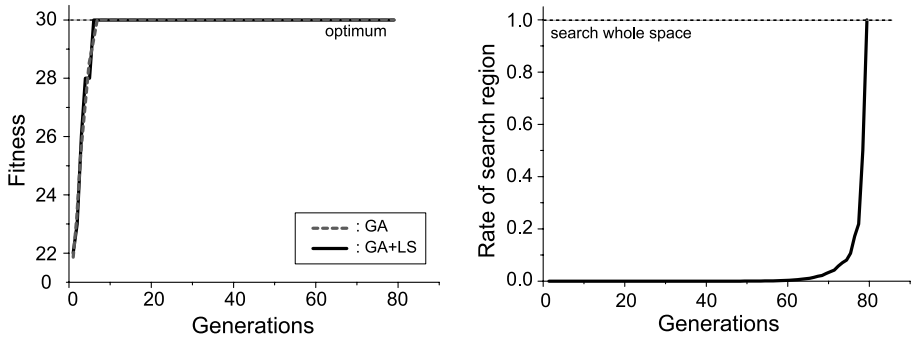
In addition, we applied our proposed method to prediction of protein tertiary structures. Proteins in nature have structures with the lowest potential energy. Therefore, their structures can be predicted by energy minimization. To treat prediction protein tertiary structures as optimization problems, energy functions that define the structures of proteins are used as objective functions, a design variable of which is the dihedral angle among the atoms that make up the proteins[19]. We compared our method to conventional GA for this problem and examined the performance of the proposed method in the distributed computing environment built using Grid MP.

### 4.1 Performance of the Proposed Method in Infinite Computation Cost

We applied the proposed method to the one max problem and 3-deceptive problem with a string length of $L$=30 without limiting computing resources. This search was terminated when all the combinations had been searched. An exhaustive search needs $2^{30}$ solutions. The ER model[20] was used as the alternation in each generation. We applied a GA with uniform crossover and each couple, or parents, generated 20 children by crossover. The mutation rate was $0.03(=1/L)$ and population size was 20. The capacity of the database, $N_{DB}$, was 5 in this experiment.

Fig. 9 shows the transition of the fitness and the ratio of the searched region on solving the one max problem. In Fig. 9(b), when the ratio of the searched region attained 1.0, all the area had been searched.

The performance of the proposed method was similar to that of the conventional GA. This was because the proposed method searched using mainly




(a) History of fitness

(b) History of the Searched Region Rate

**Fig. 9.** Performance of GA using the Local Search Mechanism in the One Max Problem

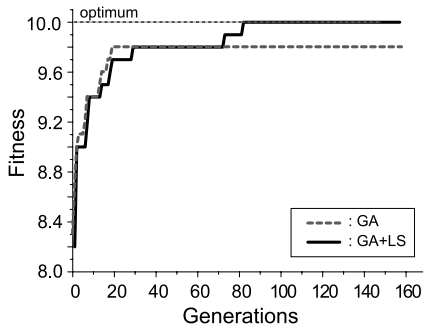
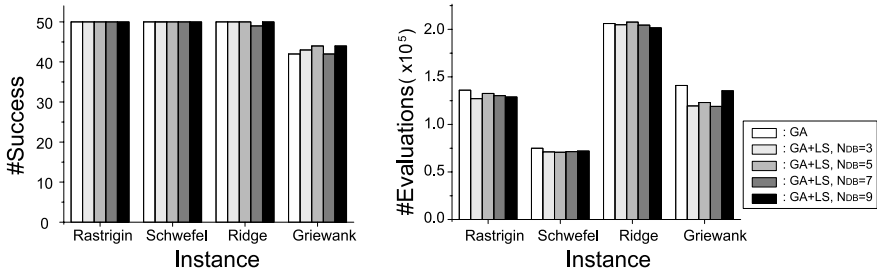



**Fig. 10.** Performance of GA using the Local Search Mechanism in the 3-deceptive Problem

schemes of GA. It is certain that the solution obtained by an exhaustive search is the optimum, although the optimal solution is obtained in the earliest part of the search. Moreover, these mechanisms enable us to show the quantitative ratio of the searched region during the search as in Fig. 9(b).

Fig. 10 shows the transition of the fitness in the 3-deceptive problem. Both the conventional GA and our proposed method fell into local optima in the early stages of the search. In the 3-deceptive problem, it is difficult for GAs to obtain the optimal solution because populations tend to be trapped by local optima. Similar to a conventional GA, our proposed method obtains convergence. Nevertheless, it can obtain the optimum as increases in computing costs yield increases in the searched regions. As a consequence of this, the optimal solution can be found by continuing the search.




(a) Number of Successes (b) Number of Evaluations

**Fig. 11.** Performance of GA using the Local Search Mechanism with Continuous Functions

### 4.2 Performance of the Proposed Method with Limited Computation Cost

To perform the exhaustive search, many evaluations of individuals are required. Our proposed method can perform an exhaustive search. It is expected that the proposed method has a high possibility of finding the optimum solution in the early stages of searches because it mainly uses schemes of GA. Therefore, we then examined whether our method can obtain the optimum under limiting evaluations.

We applied the proposed method to four test continuous functions to compare it with conventional GA. In each function, an optimum solution was attempted by the proposed method. Each function had 10 dimensions and the number of evaluations was limited to $2.5\text{x}10^5$. The ER model was used as the alternation in each generation. We applied a GA with uniform crossover and each couple, or parents, generated 20 children by crossover. We set the mutation rate to $0.01(=1/L)$ and the population size to 200. The capacity of the database, $N_{DB}$, was set to 3, 5, 7, and 9 in this experiment.

Fig. 11 describes the number of trials that obtained the optimum and the average number of evaluations needed to acquire the optimum. The results shown are from 50 trials.

Fig. 11(a) illustrates that the proposed method and the conventional GA obtained the optimal solution in all trials at Rastrigin function, Schwefel function, and Ridge function. The proposed method derived the optimal solutions several times at Griewank function. In addition, Fig. 11(b) indicates that our method could obtain the optimum with fewer evaluations than a conventional GA although our proposed method required many evaluations at the local search phase. These results indicated that our method also retained superior performance with limited computing costs. We focused attention on the effects of parameter $N_{DB}$ on the performance of our method. The numbers of successful trials of $N_{DB} = 3$ and 9 were greater than those of $N_{DB} = 5$ and 7, whereas the result of the number of evaluations was contrary in the Ridge function. In contrast, the number of successful trials of $N_{DB} = 9$ was better than those of

$N_{DB} = 3, 5$ and 7. As a result, there is no setting that can acquire a more optimal solution with fewer evaluations but the proposed method can search free from the setting of parameter $N_{DB}$.

### 4.3 Performance of the Proposed Method in Protein Tertiary Structure Energy Minimization Problems

To discuss the performance of our method in real complex problems, we applied the method to protein tertiary structure energy minimization problems. In this experiment, we used OPLS-AA/L[21, 22], which is one of the potential energy functions within the framework of classical mechanics consisting of certain energy terms with force-field parameters, and examined our method on a small protein named Met-enkephalin composed of 5 amino residues and 23 dihedral angles.

Applying this minimization problem to GA, the range of value of each dihedral angle was [-$\pi$, $\pi$). An angle is expressed strings of 6 bits, i.e., it is divided into $2^6$ equal intervals. dMSXF[23] was used as the crossover and the alternation in each generation. In all crossovers, the number of transitions, $k_{max}$, was set to 10 and the number of generating neighbor individuals of the respective step, $\mu$, was 10. Therefore, each couple, or parents, generated 100 children. We set the population size to 40, 60, and 80. The capacity of the database, $N_{DB}$, was set to 5 in this experiment. The number of evaluations was limited to 1.2x$10^5$.

Table 2 shows the best, the average, the median and the worst value of obtained energies. Our method retains the performance of a conventional GA in

**Table 2.** The Performance of Proposed Method on Met-enkephalin

| Population | GA+LS | | | | GA | | | |
|---|---|---|---|---|---|---|---|---|
| Size | best | med* | avg** | worst | best | med | avg | worst |
| 40 | -282.6 | -281.3 | -281.4 | -281.1 | -283.0 | -281.3 | -281.5 | -281.2 |
| 60 | -282.2 | -281.3 | -281.4 | -281.2 | -282.0 | -281.3 | -281.4 | -281.2 |
| 80 | -283.2 | -281.2 | -281.2 | -280.9 | -282.1 | -281.2 | -281.2 | -281.0 |

*:median, **:average

a complex problem, i.q., benchmark problems, such as the one max problem. The latest reported minimum energy of this protein is approximately -287.5 obtained by the method PSA/GAc[24]. This examination has not matured yet and needs sophistication of the parameters to obtain better solutions.

We performed this experiment in a heterogeneous distributed computing environment shown in Table 1. To implement our method in this environment, operations of the GA are executed on the User Machine. At the local search phase, expanding the searched region is executed in parallel using 40 Devices, forte01-15, libra, tiger, and le01-23. Fig. 12 illustrates the environment used for the experiments.

From Fig. 8, at least approximately 400 seconds exist as overhead. The local search is then executed on distributed environment when the size of the individ-

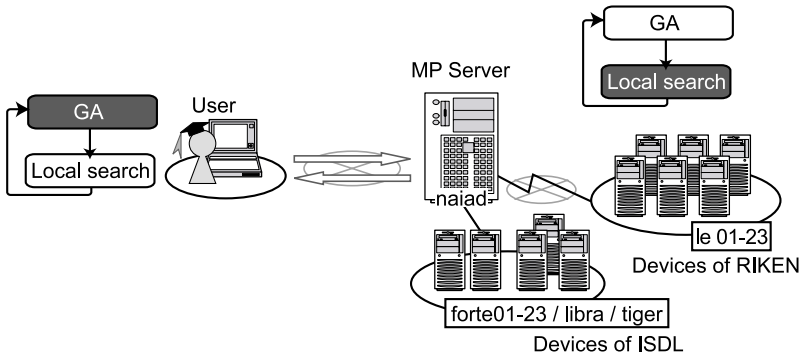



**Fig. 12.** Computing Environment for Computational Experiments

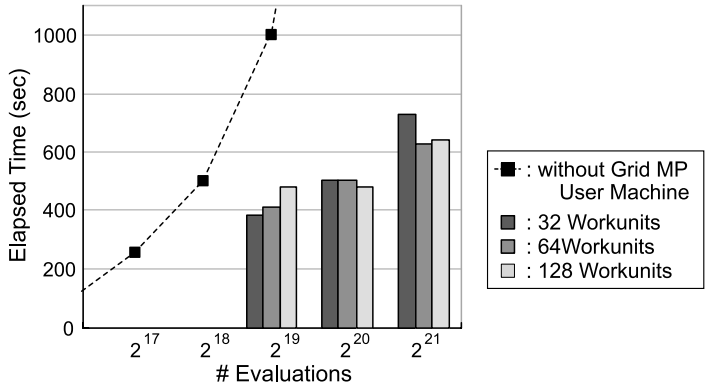


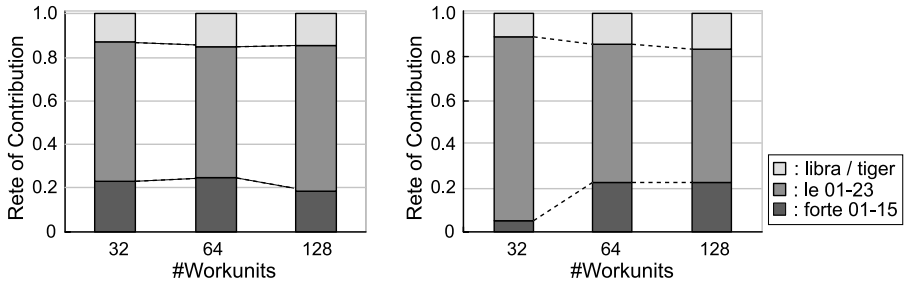

**Fig. 13.** Elapsed Times of Local Search on energy minimization of Met-enkephalin

uals that should be searched surpasses a certain value. In the energy minimization problem of Met-enkephalin, it is clear at pilot study that evaluations of $2^{18}$ individuals take approximately 500 seconds using the User Machine. Thereby evaluations in excess of $2^{19}$ should be executed in this distributed environment.

We examined the elapsed time of the Job, which evaluated a set of individuals represented with a maskstring. In this experiment, $2^{19}$, $2^{20}$, and $2^{21}$ evaluations were split to 32, 64, and 128 Workunits. In each setting of number of Workunits, the total of evaluations was divided equally to Workunits, e.g., the number of evaluations that each Workunit had was $2^{14}$ where the number of Workunits was set to 64 and total of evaluations was $2^{20}$.

Fig. 13 shows the total execution times of energy minimization problem of Met-enkephalin using Grid MP. The results are the averages of 10 trials.

It illustrates that our proposed method can execute faster on Grid MP than only using the User Machine. In addition the execution time does not depend on setting of Workunits essentially. Focusing at the result of 32 Workunits, execution time is slightly much than 64 and 128 Workunits with $2^{19}$ calculations. Grid MP




(a) $2^{20}$ calculations (b) $2^{21}$ calculations

**Fig. 14.** Contribution of Devices

has the redundancy for its high performance and fault-tolerance. Some Devices execute the same Workunit at a time and the fastest finished result is adopted when total of Workunits is less than number of Devices. As a result, the waste of calculations arises if no fault exists. In addition high spec machines will have finished calculating and sending their results before low spec machines even if low spec machines obtain Workunits earlier than high spec machines. Thus low spec machines, which are assigned Workunits that high spec machines also execute, cannot contribute their computation resources in the environment which has excessive Devices against Workunits unless high spec machines lose connections to the MP Server by some problems during executing Workunits.

Fig. 14 shows that the rate of Devices that are adopted their results, i.e., the rate of contribution of Devices, with $2^{20}$ and $2^{21}$ calculations.

The contribution was much the same in all experiments except for the result 32 Workunits with $2^{21}$ calculations. Few trials exist that forte01-15 could contribute superficially in the experiment of 32 Workunits with $2^{21}$ calculations. In this implementation, one Workunit poses $2^{16}$ calculations to a Device where the number of Workunits is set to 32 and total of evaluations is $2^{21}$. $2^{16}$ calculations are so heavy that forte01-15 cannot finish the executable even if delays of assignment of Workunits exist.

The number of Workunits has to be set bigger than the number of Devices. The overhead increases in response to increasing in Workunits. Thus appropriate number of Workunits should be set to yield high performance of proposed method in a heterogeneous distributed environment with considering contributions of Devices.

## 5 Conclusions and Future Work

GAs are suitable algorithms for parallel processing. However, increases in the number of individuals and/or computing resources do not yield improvements of performance in most methods because the diversity of the solution is also increased. Our proposed method, Tabu · Local Search mechanism for Mega Pro-

cess GA, can expand the searched region linearly as the available computing resources increases. Furthermore, the exhaustive search is guaranteed under infinite computations, while the exhaustive search is not guaranteed with conventional GAs. The proposed method was tested on the one max problem without limiting computing resources. The results confirmed that the solution obtained with this method is optimum by exhaustive search, although the optimal solution is obtained in the earliest part of the search. In addition, we applied the proposed method to four test continuous functions to derive the optimum solutions for comparison with conventional GA. These results indicated that the proposed method also retains superior performance with limited computing costs.

In addition we performed the proposed method in one of the instances of the energy minimization problems of protein tertiary structures in a heterogeneous distributed computing environment composed of 40 computation nodes belonging to RIKEN GSC and ISDL of Doshisha University, which was built up with Grid MP. We examined the execution time that included the overhead in this environment and discussed the appropriate setting of number of Workunits with considering overheads and contribution of computation nodes.

In future work, we will apply our proposed method to a large-scaled computing Grid and examine its effectiveness. Moreover, we will apply restarts in the non-searched region when the population of the GA obtains convergences as the proposed method can distinguish the non-searched region from the whole search space.

## Acknowledgments

We are grateful to Prof. Dr. Akihiko Konagaya and Fumikazu Konishi of RIKEN Genomic Sciences Center and Hiroyuki Kobayashi of Sumisho Electronics Co., Ltd.[5] for valuable discussion and contributions to the development of the distributed computing environment built using Grid MP.

## References

1. Goldberg,D.E.: Genetic Algorithms in Search Optimization and Machine Learnig. Addison-Wesley (1989)
2. H. Satoh, M. Yamamura and S. Kobayashi: Minimal Generation Gap Model for GAs Considering Both Exploration and Exploitation. Proc. of IIZUKA. pp.494-497. 1996
3. H. Kargupta: SEARCH, polynomial complexity, and the fast messy genetic algorithm. University of Illinois at Urbana-Champaign, Urbana, IL. IlliGAL Report No. 95008. 1995
4. G. R. Harik: Linkage learning in via probabilistic modeling in the ECGA. University of Illinois at Urbana-Champaign, Urbana, IL. IlliGAL Technical Report No. 99010. 1999

[5] Sumisho Electronics Co., Ltd. : http://www.sse.co.jp/e/index.html

5. I. Ono and S. Kobayashi: A Real-coded Genetic Algorithm for Function Optimization Using Unimodal Normal Distribution Crossover. Proc. of 7th Int. Conf. on Genetic Algorithms. pp.246-253. 1997
6. Pelikan,M., Goldberg,D.E., and Lobo,F.: A Survey of Optimization by Building and Using Probabilistic Models. Technical Report 99018, IlliGAL (1999)
7. Larranaga,P., Lozano,J.A.: Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation. Kluwer Academic Publishers (2001)
8. Reiko Tanese: Distributed Genetic Algorithms. Proc. 3rd International Conference on Genetic Algorithms. pp.434-439. 1989
9. T. Jansen: On the Analysis of Dynamic Restart Strategies for Evolutionary Algorithms Proc. Parallel Problem Solving from Nature - PPSN VII, 7th International Conference. pp.33-43. 2002
10. Alex S. Fukunaga: Restart Scheduling for Genetic Algorithms, Lecture Notes in Computer Science, vol.1498, pp.357-369. 1998
11. Sean Luke: When Short Runs Beat Long Runs, Proceedings of the Genetic and Evolutionary Computation Conference, pp.74-80. 2001
12. J. Maresky et al.: Selectively Destructive Restart, Proc. of Sixth International Conference on Genetic Algorithms, pp.144-150. 1995
13. Yusuke Tanimura: Parallel and Distributed Genetic Algorithm on The Cluster System and The Computational Grid. University of Doshisha. 2003, in Japanese
14. Hiroaki Imade et al.: A Grid-Oriented Genetic Algorithm for Estimating Genetic Networks by S-Systems, Proc. SICE Annual Conf. pp3317-3322, 2003
15. Hiroaki Imade et al.: A framework of grid-oriented genetic algorithms for large-scale optimization in bioinformatics Proc. of The Congress on Evolutionary Computation in Canberra. vol.1, pp623- 630, 2003
16. H. Nakata et al.: Protain structure optimizaion using Genetic Algorithm on Jojo Journal of Information Processing Society of Japan. 2002-HPC-93, pp. 155-160, 2003. in Japanese
17. United Devices. Grid MP 4.0 Application Developer's Guide, 2003.
18. Martin Pelikan et al.: BOA:The Bayesian Optimization Algorithm. IlliGAL Report No. 99003 1999
19. Y. Sakae and Y. Okamoto. Optimization of protein force-field parameters with the Protein Data bank. http://arxiv.org/abs/cond-mat/0309110.
20. D. Thierens, D. E. Goldberg: Elitist Recombination: an integrated selection recombination GA Proceedings of the 1st IEEE Conference on Evolutionary Computation pp.508-512. 1994
21. W. L. Jorgensen, D. S. Maxwell and J. Tirado-Rives. Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids. J. Am. Chem. Soc., 117, 11225-11236. 1996
22. G. A. Kaminsky, R. A. Friesner, J. Tirado-Rives and W. L. Jorgensen. Evaluation and Reparametrization of the OPLS-AA Force Field for Proteins via Comparison with Accurate Quantum Chemical Calculations on Peptides. J. Phys. Chem. B, 105, 6474-6487. 2001
23. K. Ikeda, S. Kobayashi: Deterministic Multi-step Crossover Fusion: A Handy Crossover for GAs. Proceedings of 7th International Conference on Parallel Problem Solving from Nature pp162-171. 2002
24. M. Ogura, T. Hiroyasu, M. Miki and Y. Okamoto. Implementation Models for Distributed Memory Architecture of Parallel Simulated Annealing using Genetic Crossover. Proceedings of Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems. pp121-126. 2001