Tabu · Local Search Mechanism for Mega Process Genetic Algorithm

Yoshiko Hanada
Graduate School of Engineering,
Doshisha University
1-3 Tatara Miyakodani
Kyotanabe, Kyoto 610-0321
Email: hanada@mikilab.doshisha.ac.jp

Tomoyuki Hiroyasu
Department of Engineering,
Doshisha University
1-3 Tatara Miyakodani
Kyotanabe, Kyoto 610-0321
Email: tomo@is.doshisha.ac.jp

Mitsunori Miki
Department of Engineering,
Doshisha University
1-3 Tatara Miyakodani
Kyotanabe, Kyoto 610-0321
Email: mmiki@mail.doshisha.ac.jp

***Abstract*— In this study a new Genetic Algorithm (GA) using Tabu · Local Search mechanism for large-scale computer systems is proposed. We call the GA that uses huge computing resources a Mega Process GA. The GA described in this paper is considered a Mega Process GA which has the effective mechanism to solve the problems quickly and to use massive processors, namely Mega Processors, comprised in large-scale computing systems such as super PC clusters and Grid computation environments. Our proposed method has a GA-specific database that possesses information of space that has been already searched. At the same time, the proposed GA 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 increase and the exhaustive search is guaranteed under infinite computations. Using and describing different experiments, the features of the introduced GA are discussed and examined. At first, this method was applied on one max problem and 3-deceptive problem; the former is one of primitive functions and the latter is one of trap functions. Through this experiment, it is shown that the method ensures an effective exhaustive search. This method was then applied to the test functions of continuous optimization problems under restricted computing costs. Using such an experiment, it is clear that this method has the same performance as a conventional GA.**

I. Introduction

Genetic Algorithm (GA) is one of the most effective approximation algorithms for optimization problems[1]. Various types of mechanisms are discussed for improving GAs. Minimal Generation Gap (MGG)[2] is proposed as a generation alternation model. The 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 searching ability. The restart mechanisms are also applied to enhance the performance of GA[9], [10], [11], [12].

Heretofore it is indicated that applying a GA to solve optimization problems has a drawback: GAs require lots of computing costs. One of the solutions for this problem is performing GAs in parallel. In a recent decade, owing to remarkable improvements computing abilities, some parts of the drawback about computing costs of GAs do not really matter; furthermore, parallel processing is used to yield the increase of performance of GAs. Recently, because of the emergence of super PC clusters and Grid computation environments, the number of computational calculation resources is getting larger; moreover, large computing projects of the field relating to evolutionally computing become to be feasible. GA is familiar with parallel processing, consequently the application in large-scale computing has been done[13], [14], [15], [16]; however, the adapted methods are mostly simple parallelization of conventional GAs, which are proposed for limited computing resources and the effective mechanism to use huge computing resources has not been designed. The easiest way that is commonly used for conventional GAs to use many resources is to increase the population size. However, when the population becomes large, the diversity of the solutions also becomes larger. Therefore, the convergence speed becomes slow. Subsequently when GAs use a lot of computer resources, the optimum solution is not derived quickly. Many kinds of GA methods are not optimized to use enormous computing environments effectively since they were developed within limited computing environments. At the same time, for conventional GAs, there is no guarantee of an exhaustive search on all search space although infinite computations are performed. Owing to this, though one applies simple parallelization to those methods, there is no assurance of improvement of their performance in accordance with an increased number of available computing resources.

In this study, a new GA using 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 developing the effective mechanism to use massive processors, namely Mega Processors, comprised in large-scale computing systems such as super PC clusters and Grid computation environments. The proposed method has a database that possesses information of space that is already searched. 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 a searched region during the search. Additionally using this information, the searched space increases linearly as the number of computing resources increase and an exhaustive search is guaranteed under infinite computations.

II. Database Structure

A. Representation of Searched Regions

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

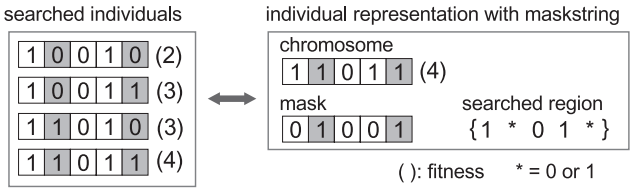


Fig. 1. An Example of an Individual stored on Database

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 or 1). We call a number of '1' in a maskstring the hamming-index, e.g. the hamming-index of x_3 is 3.

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

B. Quantitative Sizes of Searched Regions

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 a 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 $|\bigcup_{i\in I} X_i|$, where the set which consists of the suffix i is denoted $I = \{1, 2, \cdots, N\}$. In most cases, it cannot readily be derived as a function of a number of elements in a union of sets. On the other hand, that of a intersection of sets is a closed expression. The size of the searched region is such a case. Owing to this, $|\bigcup_{i\in I} X_i|$ is derived from the sets of $|\bigcap_{j\in J} X_j|$, where J is a subset of I that is shown in equation (1).

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

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).

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

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

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

III. Tabu · Local Search Mechanisms for Genetic Algorithm

A. The Concept of Proposed Method

The proposed method consists of a GA and a local search. Fig. 3 shows that the flow of our proposed method. To obtain optima earlier, our method of searches used mainly schemes of GA; any methods of operators such as a crossover and a 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 database and set to its maskstring the bitstring of which values of all loci are '0'; however, y_{best} is not stored when it is included in searched regions, which are indicated by individuals that have been already stored in the database.

Step 4. / Local search/ Expand a searched region indicated by a certain individual stored in the database. When a better

	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

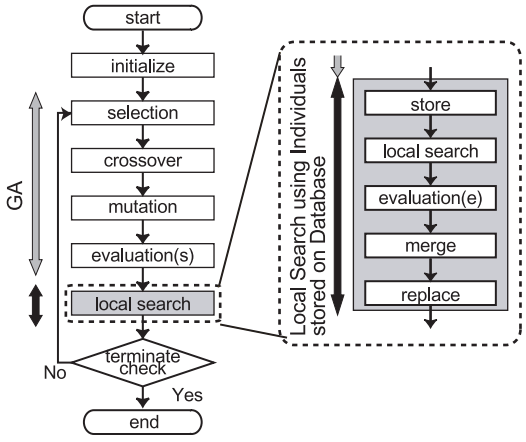


Fig. 3. The Flow of the Proposed Method

individual is found, replace the worst individual of the population of GA with it. The detail of the local search is described in the next section III-B.

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 the step 3, when there are N_{DB} individuals in the database, replace the individual x_{worst} that has the worst 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 capacity of the database.

B. Local Search

We choose 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 into a value of '1'. The locus, which holds the biggest variance of genes among whole loci, is selected. This operation is performed because it is suitable to keep much 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 the lower variances since 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 the example of the local search.

Step 1. Find the results of a_l using the 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 the 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 minimum, 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 searching 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 needs to search 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 since the searched space increases linearly as the number of computing resources increase and an exhaustive search is guaranteed under infinite computations.

C. Merge Operation in Database

Following the local search, a merge of individuals stored in the database is executed in every generation to avoid overlap-

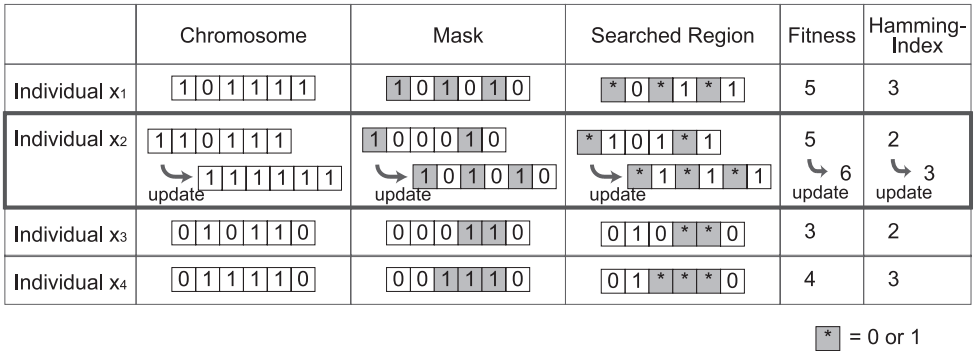


Fig. 4. An Example of Local Search on one max problem: The individual of which the search region is expanded is individual x_2. Expanding the region X_2={* 1 0 1 * 1 } to the region {* 1 * 1 * 1 } requires searching the region X_2'={* 1 1 1 * 1 }. The searched region of x_2, i.e. X_2, becomes {* 1 *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

ping 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 the 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 one 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 those 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 which is required to expand until it can include X_b. $X_a' \cap \neg X_b$ must be searched to merge using the 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 get a better solution under the available limited computing resources. This merge is not applied when A is bigger than a certain value. To decide an appropriate A, the size of search space and the number of available computing resources or costs must be considered. In this paper, we set A to eight.

IV. Numerical Experiments

To discuss the effectiveness of our proposed method both in infinite computation and limited computation, it is applied to one max problem and 3-deceptive problem[17]. The former is the most primitive benchmark problem of bitstrings, and

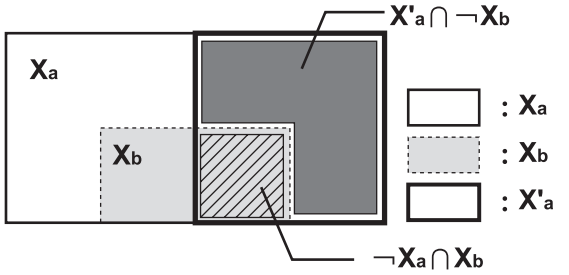


Fig. 5. Merge using Condition 3

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 under limited computation. The following equations (6), (7), (8), and (9) are the continuous test functions.

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

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

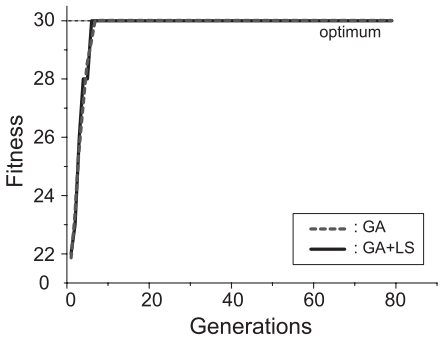
$$F_{Ridge} = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2 \tag{8}$$
$$x_i \in [-64, 64)$$

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

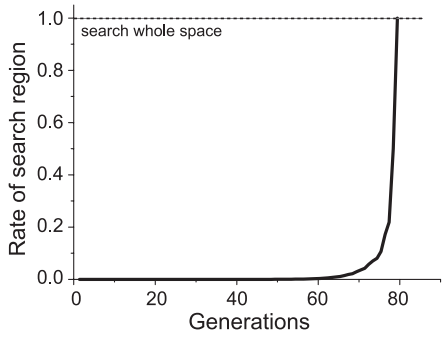
A. The Performance of Proposed Method in Infinite Computation Cost

We apply the proposed method to one max problem and 3-deceptive problem with the length of string L=30 without limiting computing resources. This search is terminated when all the combinations are searched. Exhaustive search needs 2^{30} solutions. ER model[18] is used as the alternation in each generation. We apply a GA with uniform crossover and each couple, or parents, generates 20 children by a crossover. The mutation rate is 0.03(=1/L) and population size is 20; the capacity of the database, N_{DB}, is 5 in this examination.

Fig. 6 shows the transition of the fitness and the ratio of the searched region on solving one max problem. In Fig. 6(b), when the ratio of the searched region attains 1.0, all the area has been searched.



(a) History of the Fitness



(b) History of the Searched Region Rate

Fig. 6. Performance of GA using Local · Search Mechanism on One Max Problem

The performance of the proposed method is similar to that of the conventional GA. This is because the proposed method searches using mainly schemes of GA. It is certain that an obtained solution is optimum by an exhaustive search, though 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 like Fig. 6(b).

Fig. 7 shows the transition of the fitness on 3-deceptive problem. It indicates that both conventional GA and our proposed method fall into local optima in the early stages of search. 3-deceptive problem is difficult for GAs to obtain the optimal solution because populations tend to be trapped by local optima. Our proposed method gets convergence like a conventional GA; nevertheless, it can obtain the optimum since increase of computing costs derives increase of searched regions, in consequence the optimal solution can be certainly found by continuing search.

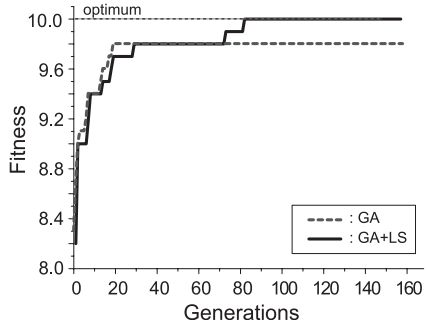


Fig. 7. Performance of GA using Local · Search Mechanism on 3-deceptive Problem

B. The Performance of Proposed Method in Limited Computation Cost

To perform the exhaustive search, many evaluations of individuals are required. Our proposed method can perform an exhaustive search; moreover, it is expected that the proposed method has high possibility to find the optimum solution in the early stages of searches because it mainly uses schemes of GA. In this section, it is then examined that our method can obtain the optimum under limiting evaluations.

We apply the proposed method to four test continuous functions to compare it with conventional GA. In each function, an optimum solution is attempted by the proposed method. Each function is 10 dimensions and the number of evaluations is limited to 2.5x10^5. ER model is used as the alternation in each generation. We apply a GA with uniform crossover and each couple, or parents, generates 20 children by a crossover. We set the mutation rate to 0.01(=1/L) and the population size to 200; the capacity of the database, N_{DB}, is set to 3, 5, 7 and 9 in this experiment.

Fig.8 and 9 describe the number of trials that obtained the optimum and the average of the number of evaluations, which were needed to acquired the optimum. These are the results of 50 trials.

Fig.8 illustrates the fact 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. Additionally, Fig.9 indicates that our method could obtain the optimum with fewer evaluations than that of a conventional GA though our proposed method requires lots of evaluations at the local search phase. These results indicate that our method also keeps its superior performance with the limited computing costs. Furthermore, we focus attention on the effect of parameter N_{DB} on the performance of our method. The numbers of successful trials of $N_{DB} = 3$ and 9 were more than those of $N_{DB} = 5$ and 7, whereas the result of the number of evaluations was contrary at Ridge function. As opposed to this, 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}.

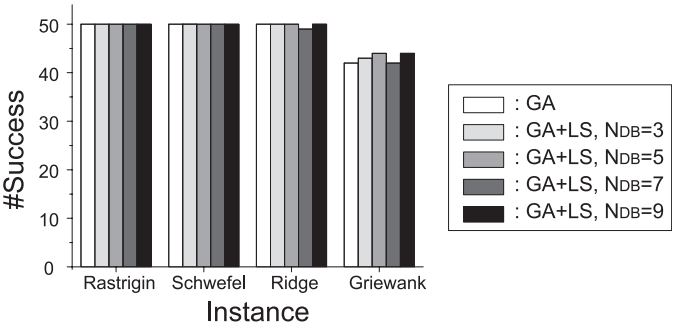


Fig. 8. Number of trials with the optimum

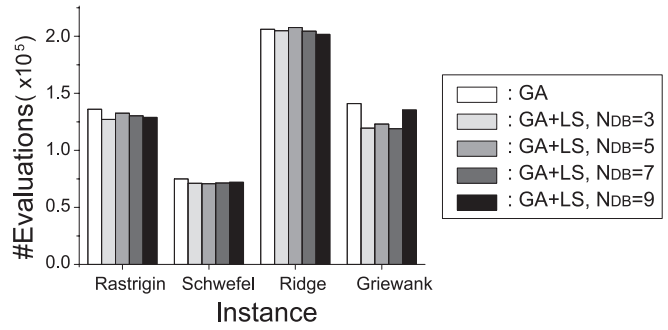


Fig. 9. Number of evaluations where method gets the optimum

V. Conclusions and Future work

GAs are suitable algorithms for parallel processing; however, there is an issue that an increase of individuals, along with an increase of computing resources, does not yield improvement of performance in most methods because the diversity of the solution is increased. Our proposed method, Tabu · Local Search mechanism for Mega Process GA, can expand the searched region linearly as the number of available computing resources increase; furthermore, the exhaustive search is guaranteed under infinite computations, while the exhaustive search is not guaranteed in the conventional GAs. The proposed method was tested on one max problem and 3-deceptive problem without limiting computing resources, as a result, it is confirmed that this method promises that an obtained solution is optimum by exhaustive search, though the optimal solution is obtained in the earliest part of the search. Additionally, we applied the proposed method to four test continuous functions for deriving the optimum solutions to compare it with conventional GA; these results indicate that the proposed method also keeps its superior performance with the limited computing costs.

For future work, we will apply our proposed method to a large-scale computing Grid and examine its effectiveness; moreover, we apply restarts at the non-searched region when the population of a GA gets convergences since the proposed method can distinguish the non-searched region from the whole search space.

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] 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] Martin Pelikan et al.: BOA:The Bayesian Optimization Algorithm. IlliGAL Report No. 99003 1999

[18] 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