Self Adaptive Island GA

Eiichi Takashima, Yoshihiro Murata, Naoki Shibata and Minoru Ito
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara 630-0192, Japan
e-mail: {eiichi-t, yosihi-m, n-sibata, ito}@is.aist-nara.ac.jp

Abstract- Exploration efficiency of GAs largely depends on parameter values. But, it is hard to manually adjust these values. To cope with this problem, several adaptive GAs which automatically adjust parameters have been proposed. However, most of the existing adaptive GAs can adapt only a few parameters at the same time. Although several adaptive GAs can adapt multiple parameters simultaneously, these algorithms require extremely large computation costs. In this paper, we propose Self Adaptive Island GA(SAIGA) which adapts four parameter values simultaneously while finding a solution to a problem. SAIGA is a kind of island GA, and it adapts parameter values using a similar mechanism to meta-GA. Throughout our evaluation experiments, we confirmed that our algorithm outperforms a simple GA using De Jong's rational parameters, and has performance close to a simple GA using manually tuned parameter values.

1 Introduction

Genetic algorithm(GA) is an approximation algorithm for combinatorial optimization problems inspired by evolution mechanisms in nature. Exploration efficiency of GAs largely depends on parameter values such as crossover rate and mutation rate. But, adjusting these parameter values takes a great deal of time, since the optimal parameter values depend on the problem to solve. Besides it, the optimal value of each parameter depends on crossover method and mutation method. To cope with this problem, several adaptive GAs which automatically adjust parameters have been proposed[1, 2, 3, 4]. However, most of the existing adaptive GAs can adapt only a few parameters at the same time. Although several adaptive GAs can adapt multiple parameters simultaneously, these algorithms require extremely large computation costs. We have already proposed an algorithm called *A-SAGA*[5] which is a combination of meta-GA[6] and GA with distributed environment scheme[7], where GA with distributed environment scheme is a kind of island GA. A-SAGA can adapt any combinations of parameters which are assigned to each island, although it only requires definitions of a fitness evaluation function and GA operators. A-SAGA requires training before solving a problem. In the training phase, many problems are solved. By using the result of training, A-SAGA solves similar problems efficiently. But, if there is only one problem to solve, A-SAGA is not always efficient since it requires extra cost of training.

In this paper, we propose Self Adaptive Island GA(SAIGA) which is based on A-SAGA and works efficiently even if there is only one problem to solve. SAIGA adapts parameter values using a similar mechanism to meta-GA, but it requires no training. Throughout our evaluation experiments, we confirmed that our algorithm outperforms a simple GA using De Jong's rational parameters[8], and has performance close to a simple GA using manually tuned parameter values. We describe related works in section 2, proposed algorithm in section 3, experimental result and consideration in section 4, and conclusion in section 5.

2 Related works

F. G. Lobo et al. have proposed an adaptive GA which effectively works when optimal number of individuals is not known[9]. In this method, parallel searches are performed using different numbers of individuals such as 16, 32, 64, and so on, expecting one or more of them with appropriate number of individuals would yield a good result. But, it is not realistic to perform a large number of searches in parallel. Accordingly, each GA is made to use a different number of evaluations per unit time so that a GA with a small number of individuals uses a larger number of evaluations per unit time. In the case where a GA with a small number of individuals can find a near optimal solution, not so much number of evaluations are wasted since GAs with a large number of individuals are only assigned a relatively small number of evaluations. In the case where only a GA with a large number of individuals can find a near optimal solution, GAs with a small number of individuals are discontinued when a GA with a larger number of individual finds a better solution.

Hinterding et al. have proposed an adaptive GA which runs three GAs with the different numbers of individuals in parallel[10]. The process of search is divided into epochs. At each epoch, fitness values of elite individuals are compared, and the number of individuals are changed according to the result. For example, if the GA with the largest number of individuals yielded the best result, all GAs will use a larger number of individuals in the next epoch.

Bäck has proposed an adaptive GA[1] whose individual has its own mutation rate encoded in its gene. Individuals with good mutation rates are expected to survive. However, since individuals with high mutation rates die in high probability, only individuals with low mutation rates tend to survive in the last phase of search. Actually, literature [11] points out that this algorithm shows only low performance in the last phase of search.

Espinoza, et al. have proposed another adaptive GA [2] whose individuals can independently search solutions using local search. In this algorithm, search efficiency per unit number of evaluations has been improved by adapting a ratio between computation costs of crossover/mutation and local search.

An adaptive GA proposed by Krink, et al.[3] determines crossover rate and mutation rate of each individual by its location in 2-dimensional lattice space. Individuals are expected to move towards a location with better parameters. The algorithm keeps diversity of these parameter values by limiting the number of individuals in each lattice.

Goldberg, et al. has proposed a theoretical way to compute effective value ranges of parameters when applying a GA to the one max problem[12, 13]. The derived ranges are depicted in a graph called *control map*. In literature [14], properties of several selection techniques have been analyzed. However, in order to derive detailed parameter values using such an analytic approach, properties of target problems must be formulated from scratch. This formulation may be difficult depending on the problem to solve.

Meta-GA is a general method which uses a GA to derive a good set of parameter values (called *parameter vector*, hereafter) used in other GAs for searching solutions. In meta-GA, since the number of evaluations tends to be large, the whole computation costs must also be high. For example, an ordinary GA with 100 individuals and 100 generations requires 100×100 evaluations. To find a good parameter vector for this GA using a meta-GA with 10 individuals and 20 generations, the number of evaluations will be $10 \times 20 \times 100 \times 100 = 2,000,000$.

Kee, et al. has improved meta-GA methods [4]. In this method, a preliminary search is carried out for training before applying the algorithm to the actual problem. In the training phase, states of individuals are classified into several groups depending on given indices. Then, for each group, search efficiency is investigated for several tens of parameter vectors and the parameter vector with the highest search efficiency is determined. When searching a solution to the actual problem, it observes the state of individuals and uses the parameter vector with the highest efficiency for that state.

Island GA (IGA)[15] is a kind of parallel GAs. In IGA, each GA is regarded as an island, and all islands are executed in parallel. Some individuals immigrate to another islands as the search progresses and in this way islands share some information and search the solution in cooperation.

Tongchim et al. have proposed an adaptive GA which adapts mutation rate and crossover rate[16]. This GA is based on IGA. As the search progresses, increases in average fitness of each island are compared to those of neighbor islands. If an increase of a neighbor island is larger, parameter values are changed according to that of the neighbor island.

Miki et al. have proposed a GA with distributed environment scheme[7]. In this GA, different parameter vectors are given to islands of IGA expecting good solutions to be found in islands with good parameter vectors. This is not an adaptive GA, since parameters have to be given manually.

3 Proposed algorithm

3.1 Overview

SAIGA uses two layers of GAs. The lower layer is an island GA called low level GA which consists of two or more islands. The set of islands in the low level GA is denoted by I. Each island in I searches for the solution to a given problem. We place these islands in ring topology. The upper layer is a simple GA called high level GA which searches for a suitable parameter vector for the low level GA.

The parameter vector corresponding to an island i in I is $\mathbf{v}_i = (n_i, s_i, c_i, m_i)$, where n_i, s_i, c_i and m_i denote population size, tournament size, crossover rate and mutation rate respectively.

Our algorithm executes one generation of the high level GA at every predefined number of evaluations of the low level GA. We call these predefined number of evaluations *era*. Islands independently search for a better solution using predefined number of evaluations at each era. For example, if 100 evaluations are assigned to each island, and $n_i = 20$ is assigned to one of the islands, $\frac{100}{20} = 5$ generations of search is performed during that era.

After each era, fitness values are determined for all parameter vectors. In meta-GA, parameter vectors are evaluated by individual fitness of the elite individual. But, this evaluation method does not work well with our algorithm, since when one of the immigrants has better fitness value than the original elite individual, the fitness value of the elite individual increases. Accordingly, parameter fitness of our algorithm is given by cumulative increases of individual fitness of elite individual, except increase caused by immigration.

Strictly speaking, if populations of two islands are different, the best parameter vectors for these islands might also be different. But, since the islands share information by immigration, we regard that the best parameter vectors for these islands are same.

Using our algorithm, we need not adjust the parameter vector for the low level GA, but still need to adjust the parameter values for the high level GA. But, there should be a parameter vector suitable for any kind of problems, since the high level GA always solves a problem to find the best parameter vector for the low level GA, and there is not so much difference between these problems unless the set of parameters are changed, even if the tasks for the low level GA are different.

3.2 Detailed explanation

3.2.1 Encoding of parameter vector for low level GA

n'_i, s'_i, c'_i and m'_i denote genotypes for population size, tournament size, crossover rate and mutation rate, respectively(figure 1). Each of the genotypes is represented by a 16bit fixed point number whose range is between 0 and 1. The genotypes are converted to corresponding phenotypes n_i, s_i, c_i and m_i using the following formulas.

$$n_i = \lfloor 2 \cdot \exp(8 \cdot n'_i \cdot \log(2)) \rfloor . \quad (1)$$

$$s_i = \begin{cases} \lfloor n_i \cdot s'_i \rfloor & \text{if } n_i \cdot s'_i > 2, \\ 2 & \text{if } n_i \cdot s'_i \leq 2. \end{cases} \quad (2)$$

$$c_i = c'_i . \quad (3)$$

$$m_i = 0.00005 \cdot \exp(m'_i \cdot \log(1/0.0001)). \quad (4)$$

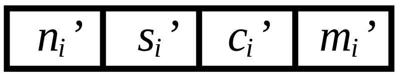

Figure 1: Encoding of parameter vector for low level GA

3.2.2 Crossover and mutation methods for the high level GA

We use uniform crossover as the crossover method for the high level GA, provided that both of the target individuals have population size more than 2. Otherwise, n'_i and s'_i are handled as one gene and not separated. This is because when n_i is 2 or less, s_i is 2 regardless of s'_i and thus s'_i is not evaluated properly.

The mutation operator of the high level GA is as follows.

Mutation operator are applied to every gene. For example, mutation operator applied to population size is defined as

$$n'_i \leftarrow n'_i + N(0, 0.1),$$

where $N(0, 0.1)$ is a gauss random number with mean 0 and variance 0.1.

3.2.3 Pseudo code

We use the following constants in our algorithm.

max_era_count	2500
evaluation_count_per_era	256
high_level_crossover_rate	0.8
high_level_mutation_rate	0.6
$\lvert I \rvert$ (the number of islands)	10

We use the following notation in out algorithm.

P^h is the set of individuals in the high level GA. $p^h[i]$ ($\in P^h$) is a high level individual for island i. Each island i in I has $\mathrm{evaluation\_count\_per\_era}$ individuals. $p^l[i][1], ..., p^l[i][\mathrm{evaluation\_count\_per\_era}]$ are global variables, where $p^l[i][k]$ is the k-th individual in island i. P^l_i is a set of individuals in the island i. B_i is a buffer for receiving immigrants to island i. B_i is global variable. f_i will retain elite fitness value of island i. ec retains the number of era count. g_i retains fitness value of the high level individual corresponding to island i. evc retains the number of evaluation performed in this subroutine. f_{inc} retains increase of the elite fitness. f_{current} retains the elite fitness in the next generation. o_{best} is the elite individual.

Pseudo code of the proposed algorithm is as follows.

Algorithm *SAIGA*

```
begin
for each $i \in I$ do
    randomly generate $p^h[i]$;
    randomly generate $p^l[i][1], ..., p^l[i][\mathrm{evaluation\_count\_per\_era}]$;
    $B_i$ := $\emptyset$;
    $f_i$ := 0;
next
for $ec$ := 0 to max_era_count do
    // Each individual for high level GA corresponds to an island.
    for each $i \in I$ do
        $(P^l_i, g_i, f_i)$ := Executeisland($P^l_i$,$p^h[i]$,$f_i$,$i$);/* Execute one era of island. */
    next
    Perform roulette selection of individuals for high level GA using fitness value $g_i$. The elite individual is conserved.
    $P^h$ := HighLevelCrossover($P^h$);
    $P^h$ := HighLevelMutation($P^h$);
next
end
```

Algorithm *Executeisland(P^l_i, $p^h[i]$,f_{last},i)*

```
begin
$evc$ := 0;
$f_{\mathrm{inc}}$ := 0;
Let $(n, s, c, m)$ be phenotypes of $p^h[i]$.
while $evc$ < evaluation_count_per_era do
    Perform crossover to $P^l_i$ with probability $c$ per individual. The elite individual is preserved.
```

```
7       Perform mutation to $P_i^l$ with probability $m$ per individual.
        The elite individual is preserved.
8       Substitute a sufficiently small number for $f_{\mathrm{current}}$.
9       for each newly generated individual $o \in P_i^l$ do
10          Calculate fitness value of $o$ and substitute this value
            for $f_o$
11          $evc$ := $evc$ + 1;
12          if $f_{\mathrm{current}} < f_o$ then
13              $f_{\mathrm{current}}$ := $f_o$;
14              $o_{\mathrm{best}}$ := $o$;
15          endif
16      next
17      Perform tournament selection with size $s$ to $P_i^l$ based on
        each fitness value $f_o$, and let $\{p^l[i][1], ..., p^l[i][n]\}$ be re-
        sult of the selection. The elite individual is preserved.
18      $f_{\mathrm{inc}}$ := $f_{\mathrm{inc}} + f_{\mathrm{current}} - f_{\mathrm{last}}$;
19      $f_{\mathrm{last}}$ := $f_{\mathrm{current}}$;
20  end while
    /*Process emigration*/
21  $d$ := emigration_destination($i$); /* Function emigra-
    tion_destination($i$) returns emigration destination from island
    $i$. islands are placed in a ring topology.*/
22  $B_d$ := $B_d \cup \{o_{\mathrm{best}}\}$
    /*Receive immigrants*/
23  if $B_i \neq \emptyset$ then
24      Substitute $e$ for the individual with highest fitness in $B_i$.
25      if the fitness value of $e > f_{\mathrm{last}}$ then
26          $f_{\mathrm{last}}$ := the fitness value of $e$;
27          Choose an individual from $p^l[i][1]$ to $p^l[i][n]$ except
            the elite individual, and substitute it for $e$.
28      endif
29  endif
30  return ($P_i^l, f_{\mathrm{inc}}, f_{\mathrm{last}}$);
31  end
```

Algorithm *HighLevelCrossover(P^h)*

```
1   begin
2   for i := 1 to $0.5 \cdot |I| \cdot$ high_level_crossover_rate do
3       Generate a uniform random number r, where $0 \leq r \leq 1$.
4       if $r < 0.5$ then
5           swap($n'_{2i}, n'_{2i+1}$);
6           swap($s'_{2i}, s'_{2i+1}$);
7       endif
8       if $n_{2i} > 2$ and $n_{2i+1} > 2$ then
9           Generate a uniform random number r, where $0 \leq r \leq$
            1.
10          if $r < 0.5$ then swap($s'_{2i}, s'_{2i+1}$); endif
11      endif
12      Generate a uniform random number r, where $0 \leq r \leq 1$.
13      if $r < 0.5$ then swap($c'_{2i}, c'_{2i+1}$); endif
14      Generate a uniform random number r, where $0 \leq r \leq 1$.
15      if $r < 0.5$ then swap($m'_{2i}, m'_{2i+1}$); endif
        $(n'_{2i+1}, s'_{2i+1}, c'_{2i+1}, m'_{2i+1})$.
16  next
17  end
```

Algorithm *HighLevelMutation(P^h)*

```
1   Begin
2   for each $p^h[i] \in P^h$ do
3       Let $(n', s', c', m')$ be genotypes of $p^h[i]$.
4       Generate a uniform random number r, where $0 \leq r \leq 1$.
5       if $r <$ high_level_mutation_rate then
6           $n' := n' + N(0, 0.1)$;
            /*Function $N(0, 0.1)$ returns a gauss random number
            with mean 0 and variance 0.1*/
7           if $n' < 0$ then $n'$ := 0; endif
8           if $n' > 1$ then $n'$ := 1; endif
9       endif
10      Generate a uniform random number r, where $0 \leq r \leq 1$.
11      if $r <$ high_level_mutation_rate then
12          $s' := s' + N(0, 0.1)$;
13          if $s' < 0$ then $s'$ := 0; endif
14          if $s' > 1$ then $s'$ := 1; endif
15          $s := \lfloor n \cdot s' \rfloor$;
16          if $s < 2$ then $s := 2$; endif
17      endif
18      Generate a uniform random number r, where $0 \leq r \leq 1$.
19      if $r <$ high_level_mutation_rate then
20          $c' := c' + N(0, 0.1)$;
21          if $c' < 0$ then $c'$ := 0; endif
22          if $c' > 1$ then $c'$ := 1; endif
23      endif
24      Generate a uniform random number r, where $0 \leq r \leq 1$.
25      if $r <$ high_level_mutation_rate then
26          $m' := m' + N(0, 0.1)$;
27          if $m' < 0$ then $m'$ := 0; endif
28          if $m' > 1$ then $m'$ := 1; endif
29      endif
30      Let $p^h[i]$ be a chromosome with genotypes
        $(n', s', c', m')$.
31  next
32  end
```

4 Evaluation

4.1 Overview

For evaluation, we apply our algorithm to Traveling Salesman Problem (TSP), deceptive problem, minimization problems of Rastrigin function and Griewank function. We compare our algorithm to a simple GA using De Jong's rational parameters [8], where mutation rate is 0.001, population size is 50, crossover rate is 0.6 and tournament size is 2. We also compare our algorithm to a simple GA using manually tuned parameter values.

4.2 Problems

4.2.1 Minimization problem of Rastrigin and Griewank function

We use Rastrigin(equation 5) and Griewank function (equation 6) with 30 variables where each variable is encoded by Gray code. We use one point crossover as the low level

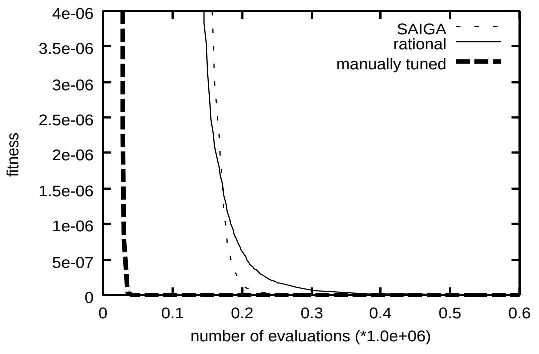

Figure 2: Search efficiency on minimization problem of Rastrigin function

crossover operator and bit reverse as the low level mutation operator.

$$f(x) = 3.0 \cdot N + \sum_{i=1}^{N} \left(x_i^2 - 3.0 \cdot \cos(2\pi x_i)\right), \quad (5)$$
$$-2.048 \le x_i < 2.048.$$

$$f(x) = 1 + \sum_{i=1}^{N} \left(\frac{x_i^2}{4000}\right) - \prod_{i=1}^{N} \left(\cos\left(\frac{x_i}{\sqrt{i}}\right)\right), \quad (6)$$
$$-512 \le x_i < 512.$$

4.2.2 Deceptive problem

We use the problem which is a concatenation of 4 deceptive functions with an 8bit variable each, shown as expression 7 where $g(u)$ is the number of 1's in u expressed in binary form. Each variable is encoded in binary.

$$f(g(u)) = \begin{cases} 0 & \text{if } g(u) = 0, \\ 9 - g(u) & \text{if } g(u) \neq 0. \end{cases} \quad (7)$$

We use one point crossover as the low level crossover operator and bit reverse as the low level mutation operator.

4.2.3 Traveling salesperson problem

We apply our algorithm to lin 105 problem[17] which has 105 cities. The optimal solution to this problem is 14379. We use 2opt method as a mutation operator. Whether the mutation operator is applied or not is determined for each city. We use EXX[18] as a crossover operator.

4.3 Search efficiency

The results for search efficiency of TSP, deceptive problem, the minimization problems of Rastrigin function and Griewank function are shown in figure 2 , 3 , 4 and 5 respectively.

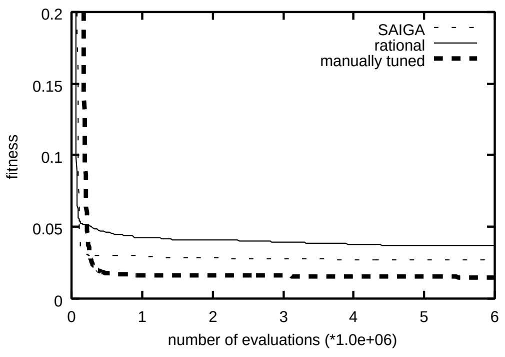

Figure 3: Search efficiency on minimization problem of Griewank function

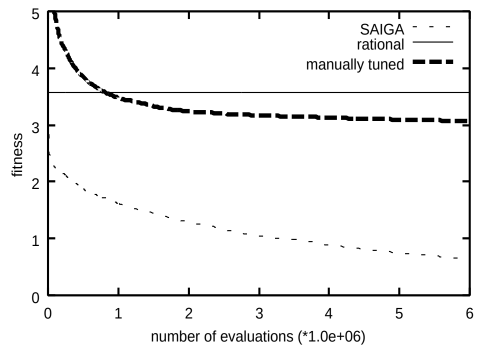

Figure 4: Search efficiency on deceptive problem

The vertical axis represents fitness value of the elite individual, and the horizontal axis represents the number of evaluations. Each of the results is the average of 300 trials.

At the beginning of the search, our algorithm tends to be outperformed by a simple GA using the rational parameters, but eventually our algorithm outperforms the simple GA. This is because our algorithm uses randomly initialized parameter values at first and thus its search efficiency is low, but search efficiency becomes high as the parameter vector converges towards the optimal value. Also, our algorithm has performance close to a simple GA using manually tuned parameter values. Similarly, the performance of our algorithm is close to that of a simple GA using manually tuned parameters on TSP, the minimization problems of Rastrigin function and Griewank function.

On the deceptive problem, both of the simple GAs stick after finding a local optima whose fitness value is 4. We can see that the average of the fitness values after convergence is a little less than 4. This is because we use the problem which is a concatenation of 4 independent deceptive problems, and rarely optimal solutions are found for some of these problems. Our algorithm escapes the local optima, since it has

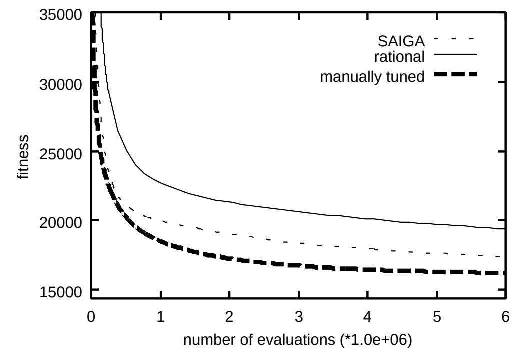

Figure 5: Search efficiency on TSP

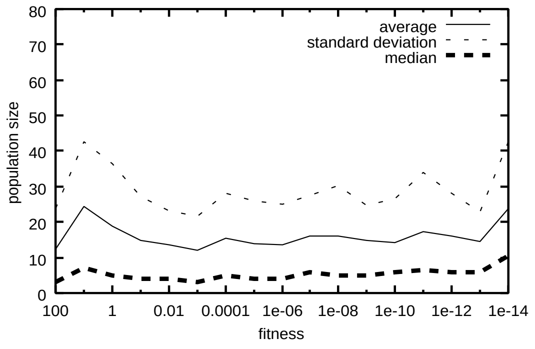

Figure 6: Transition of population size n on Rastrigin function

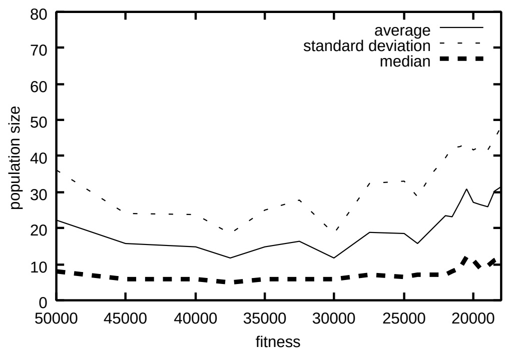

Figure 7: Transition of population size n on TSP

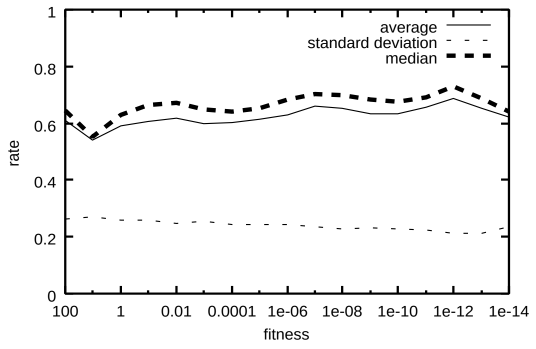

Figure 8: Transition of crossover rate c on Rastrigin function

the characteristic of IGA.

As shown in figure 2, the performance of our algorithm when applied to the minimization problem of Rastrigin function is similar to a simple GA using the rational parameters.

4.4 Transition of adapted parameter values

Processes of parameter adaptations for each problem are contrasted as follows. As of the problem of Rastrigin function, population size, mutation rate, crossover rate and tournament size are shown in figure 6, 8, 10 and 12 respectively. As of TSP, population size, mutation rate, crossover rate and tournament size are shown in figure 7, 9, 11 and 13 respectively. The vertical axis represents each parameter value, and the horizontal axis represents fitness value. In these figures, average, median and standard deviation of elite parameter value of 300 trials are shown.

We can see that population size converges to different values for each problem. Thus, we consider that SAIGA adapts the population size to each problem suitably. This seems to be because there is difference between degrees of diversity required for each problem. As diversity in the population becomes little, the population is likely to stick in a local optima, although the efficiency of local search becomes high.

As of TSP, the average population size decreases to 15 momentary, but the average elite population size when fitness value is around 20000 is larger than 30, and this seems to be because our algorithm is trying to increasing diversity.

As of the minimization problem of Rastrigin function, population size decreases on the early stages of the search. This seems to be because local search is effective on the early stages of the search. This is also seen in the transition of parameters on the problem of Griewank function.

As of the problem of Rastrigin function, mutation rate decreases as the search proceeds. This seems to be because it finds a better solution as the search proceeds, and local search becomes effective because of the property of the problem. In contrast, as of TSP, mutation rate does not change from 0.02. With this rate, two 2opt operations are expected to be applied for one individual in one low level generation.

As of TSP, crossover rate does not change at the beginning of search. But, it gradually increases as the search proceeds. This seems to be because other parameter values have larger influences on search efficiency at the beginning, but influences of crossover rate becomes high at a later stage. We consider that SAIGA successfully adapts crossover rate. As of the problem of Rastrigin function,

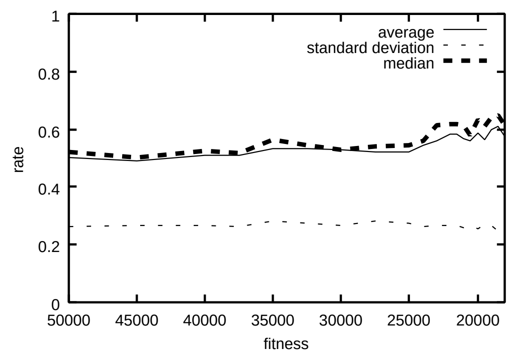

Figure 9: Transition of crossover rate c on TSP

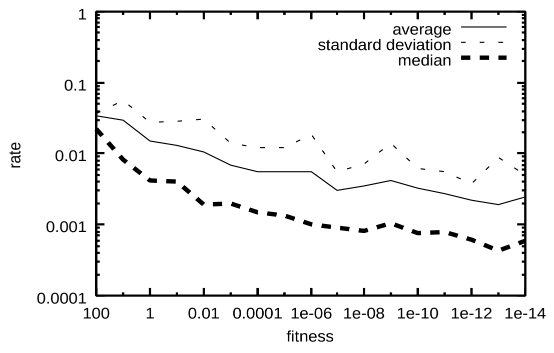

Figure 10: Transition of mutation rate m on Rastrigin function

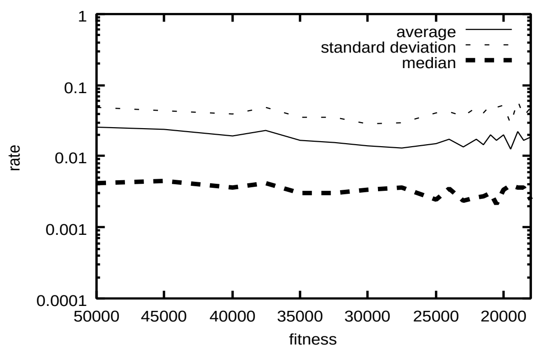

Figure 11: Transition of mutation rate m on TSP

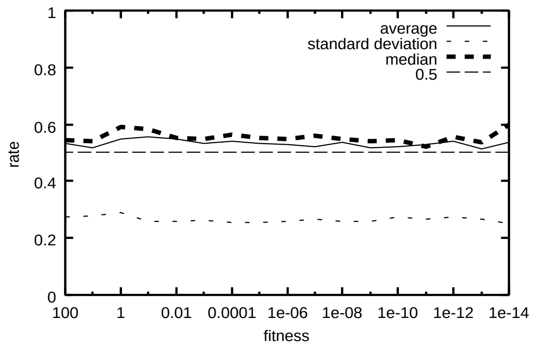

Figure 12: Transition of rate of tournament size s' on Rastrigin function

crossover rate is always around 0.6. But, we consider that SAIGA successfully adapts crossover rate since if it fails and crossover rate changes randomly, the average should be 0.5.

As of TSP and the problem of Rastrigin function, tournament size ratio s'_i does not change from around 0.5. This value is close to the average value where s'_i randomly decided, and this might suggest that our algorithm would fail to adapt the tournament size. However, we observed transition of s'_i when s'_i is initialized to 1.0e-5, and s'_i converged to 0.5 after fitness value becomes 21500 or lower. Thus, we consider that SAIGA successfully adapts tournament size.

On figure 14, distributions of adapted values of s'_i are given when fitness values are around 40000 and 19500. We can see that when fitness value is around 40000, the value of s'_i is mostly 0.7 to 0.8, and when fitness value is around 19500, the value of s'_i is mostly 0.2 to 0.3. This seems to be because search efficiency becomes high when selection pressure is high at the beginning of search. But diversity of population is required after the fitness value becomes less than 25000, and thus tournament size decreases.

5 Conclusions

We proposed a self adaptive Island GA which adapts 4 parameter values simultaneously by giving different parameter values to each island and observing search efficiencies. Throughout our evaluation experiments, we confirmed that our algorithm outperforms a simple GA using De Jong's rational parameters, and has performance close to a simple GA using manually tuned parameter values. Also, we confirmed that SAIGA successfully adapts each parameter values.

In the future work, we would like to improve handling of parameter values for the low level GA when the low level GA is stuck in a local optima.

Bibliography

[1] Bäck, T. Self-adaptation in genetic algorithms. In F.J.Varela, P. B., editor, Proceedings of 1st European Conference on Artificial Life, pp. 263–271, (1992).

[2] Espinoza, F., Minsker, B. S. and Goldberg, D. A Self-Adaptive Hybrid Genetic Algorithm. Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, Morgan Kaufmann Publishers, p. 759, (2001).

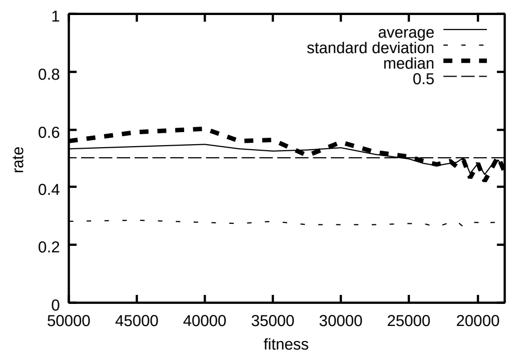

Figure 13: Transition of rate of tournament size s'on TSP

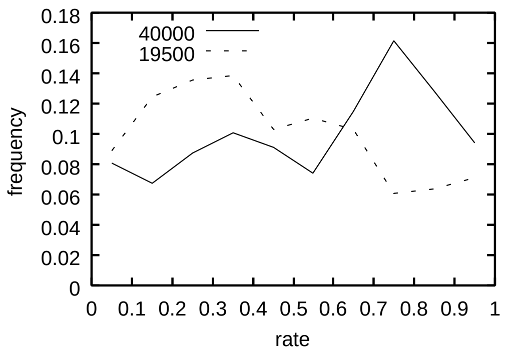

Figure 14: Distribution of adapted value of rate of tournament size s' on TSP

[3] Krink, T. and Ursem, R. K. Parameter Control Using the Agent Based Patchwork Model. Proceedings of the Congress on Evolutionary Computation, pp. 77–83, (2000).

[4] Kee, E., Airey, S. and Cye, W. An Adaptive Genetic Algorithm. Proceedings of the Genetic and Evolutionary Computation Conference, pp. 391–397, (2001).

[5] Murata, Y. Shibata, N. Yasumoto, K. and Ito, M. Agent Oriented Self Adaptive Genetic Algorithm. Communications and computer networks(CCN). November 4-6, 2002 cambridge, USA, pp. 348–353, (2002).

[6] Weinberg, R. Computer simulation of a living cell. Dissertations Abstracts International, 31(9), 5312B, (1970).

[7] Miki, M. Hiroyasu, K. Kaneko, M. and Hatanaka, I. A Parallel Genetic Algorithm with Distributed Environment Scheme. IEEE Proceedings of Systems, Man and Cybernetics Conference SMC'99, pp. 695–700, (1999).

[8] De Jong, K, A. An analysis of the behavior of a class of genetic adaptive systems. Ph. D. University of Michigan, Ann Arbor, (1975).

[9] F. G. Lobo. The Parameter-Less Genetic Algorithm:Rational and Automated Parameter Selection for Simplified Genetic Algorithm Operation. PhD thesis, University of Lisbon, Portugal, (2000).

[10] Hinterding, R., Michalewicz, Z. and T. C. Peachey. R. Hinterding, Z. Michalewicz, and T. C. Peachey. Self-adaptive genetic algorithm for numeric functions. Proceedings of the 4th Conference on Parallel Problem Solving from Nature, pp. 420–429, (1996).

[11] Glicman, M. R. and Sycara, K. Reasons for Premature Convergence of Self-Adapting Mutation Rates. Proceedings of the Congress on Evolutionary Computation, pp. 62–69, (2000).

[12] Goldberg, D. Sizing populations for serial and parallel genetic algorithms, In J. Davis Schaffer, editor, Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, pp. 70–79, (1989).

[13] Goldberg, D., Deb, K. and Dirk, T. Toward a better understanding of mixing in genetic algorithms. Journal of the Society of Instrument and Control Engineers, 32(1):pp. 10–16, (1993).

[14] Goldberg, D. and Deb, K. A comparative analysis of selection schemes used in genetic algorithms. In Gregory J.E. Rawlins, editor, Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, pp. 69–93, (1991).

[15] Tanese, R. Distributed Genetic Algorithms. In Proceedings of the Third International Conference on Genetic Algorithms, pp. 434–439, (1989).

[16] Tongchim, S. and Chongstitvatana, P. Parallel genetic algorithm with parameter adaptation, Information Processing Letters, Volume 82, Issue1, pp. 47–54, (2002).

[17] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

[18] Maekawa, K. Mori, N. Tamaki, H. Kita H. and Nishikawa, H. Genetic Solution for the Traveling Salesman Problem by Means of a Thermodynamical Selection Rule. Proceedings 1996 IEEE International Conference on Evolutionary Computation, pp.529–534, (1996).