Agent Oriented Self Adaptive Genetic Algorithm

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

ABSTRACT
Efficiency of Genetic Algorithms (GAs) depends largely on the parameters such as crossover rate and mutation rate. In general, however, it is difficult to adjust those parameters manually. Although there are a few researches about adaptive GAs for adjusting multiple parameters, they require extremely large computation costs. In this paper, we propose a new algorithm based on multi agent techniques which combines existing meta-GA techniques and GAs with distributed environment schemes.

Through some simulations, we have confirmed that the proposed algorithm can adapt multiple parameters in reasonable computation costs.

KEY WORDS
Software Agents, Artificial Intelligence Applications, Modelling and Simulation

1 Introduction

Methods to solve combinatorial optimization problems with computers are especially important among many research areas, and a lot of work has been done so far. Since the problems belong to the NP-complete class [6], it would be almost impossible to calculate optimal solutions for large scale problems within practical time. Therefore, some approximation techniques have been needed to calculate semi-optimal solutions efficiently. Among them, genetic algorithms (GAs) [5] has drawn much attention.

GAs is based on evolution mechanisms of creatures, and is expected as a promising method which can be applied to wide areas of problems. However, efficiency of GAs depends largely on the parameters such as crossover rate and mutation rate. In general, it is difficult to adjust those parameters manually. To cope with this problem, several GAs for automatically adjusting parameters have been proposed[1, 3, 7, 8]. However, most of existing techniques can adjust only a few parameters. Although a few researchers have proposed adaptive GAs which adjust multiple parameters, they require extremely large computation costs.

In this paper, we propose a new algorithm called *A-SAGA* (Agent oriented Self Adaptive Genetic Algorithm) which combines existing meta-GA techniques and GA with distributed environment scheme. Our algorithm can adjust parameters while exploring solutions. Through some simulations, we have confirmed that the proposed algorithm can be adapted for four parameters within reasonable computation costs.

2 Related Works

Bäck has proposed an adaptive GA[1] where each individual has its own mutation rate encoded in its gene so that individuals with good mutation rates are expected to survive. However, since individuals with high mutation rates may also die in high probability, individuals with low mutation rates tend to survive in the near last phase of GA exploration. Actually, in [4] it is reported that this algorithm shows low performance in the last phase.

Espinoza, et al. have proposed another adaptive GA [3] where individuals can explore solutions in local search independently of each other. In the algorithm, exploration efficiency per evaluation has been improved by automatically adjusting ratio between computation costs of crossover/mutation and local search.

In the adaptive GA proposed by Krink, et al.[8], the crossover rate and the mutation rate are determined depending on locations of individuals in 2-dimensional lattice space so that each individual moves towards the location with better parameters. The algorithm makes it possible to keep diversity of the crossover rate and the mutation rate by limiting the number of individuals which can exist in each lattice of the space.

The meta-GA is a general method which uses GAs to derive good initial values of parameters used in another GAs. In the meta-GA, since the number of evaluations tends to become large, the whole computation costs also become high. For example, the number of evaluations will usually be 100×100 in GA exploration with 100 individuals and 100 generations. Also, in the meta-GA, since each evaluation is done by executing GA one time, when executing meta-GA with 10 individuals and 20 generations, the number of evaluations will be $10 \times 20 \times 100 \times 100 = 2000000$.

Kee, et al. has improved meta-GA methods in [7]. In the improved method, preliminary exploration is carried out for training before applying the algorithm to the actual problem. Then, it examines what combinations of parameter values have higher exploration efficiency among sev-

eral tens of combinations of parameter values. In exploring the actual problem, it dynamically selects one combination with the highest efficiency, depending on the states of individuals.

The island GA (IGA)[2] is known as one of parallel GAs. In the GA with distributed environment scheme which has been proposed by Miki, et al. in [9], different parameter values are given to islands of IGA, respectively. By this, it is expected that better solutions can be found in islands with good parameter values.

3 Proposed algorithm

3.1 Problems in existing methods and our solution

As we explained in Sect. 2, most of existing adaptive GAs can adapt only one or two parameters. Few GAs can adapt multiple parameters at the same time. Although the meta-GA can adapt multiple parameters simultaneously, it requires large computation costs.

The reason why the meta-GA requires large computation costs is that we must run GA exploration once (we call this *meta-GA generation*) to evaluate one parameter vector.

As the number of parameter vectors evaluated by meta-GA becomes high, it will be able to find a good parameter vector in high probability.

Suppose that the total computation costs are limited to a certain extent. In such a condition, if we want to evaluate parameter vectors as many as possible, we must reduce computation costs during each GA phase. However, when reducing computation costs during GA phase, it would be difficult for a GA to find a good solution due to insufficient exploration in the solution domain. Moreover, in general, good parameter vectors for GAs with reduced computation costs are not always good for GAs with sufficient ones.

3.2 Introduction of agent oriented methods

For the above problems, we developed a preliminary algorithm called A-SAGAβ which is a hybrid algorithm consisting of GAs with distributed environment scheme[9] and the meta-GA. In A-SAGAβ, we use multiple agents where we let each agent run a GA with reduced computation costs, and give different parameter vectors to agents, respectively. Those agents explore a solution in cooperation with each other by exchanging immigrants.

Since this system can be regarded as one of GA with distributed environment scheme, we can obtain rather a good solution as the whole system even when the computation costs of each agent are restricted.

As described above, the best parameter vectors of GAs with reduced computation costs differ from those of GAs with sufficient computation costs in general. Thus, if we try to reduce computation costs of existing meta-GA, we have to reduce the number to run GAs exploration and this leads to insufficient explore of parameter vectors. With our method, parameter vectors whose number is the same as the number of agents can be obtained with one meta-GA generation, and thus computation costs of parameter adaptation can be reduced.

1. initialize all individuals.
2. evaluate individuals.
3. select some individuals.
4. crossover the selected individuals.
5. mutate individuals.
6. if cumulative evaluation times exceed a threshold, send some individuals to other agents as immigrants.
7. otherwise, go to step 2.

Figure 1. A-SAGAβ algorithm in each agent

1. initialize all agents
2. explore a solution by each agent
3. select some agents
4. crossover the selected agents
5. mutate agents
6. explore a solution by each agent
7. if repetition times are below a threshold, go to step 3.

Figure 2. A-SAGAβ algorithm as the whole system

In multi agent systems, however, social welfare in the whole system is not always maximized when greedy agents maximize their benefits. Therefore, if all of agents in a system use a parameter vector which suits one of the agents, total performance of the system is not always improved. This is a general problem in the multi agent system.

This problem tends to be prominent in such situation that roles of agents are not symmetric. In A-SAGAβ, agents are placed in ring topology so that all of agents are symmetric[2].

3.3 Preliminary algorithm

In A-SAGAβ, a parameter vector $\mathbf{v}_\beta = (n, p_\mathrm{m}, p_\mathrm{c}, l)$ is given to each agent. Here, n is the number of individuals, p_m and p_c denote the mutation rate and the crossover rate, respectively, and l denotes the linear scaling coefficient.

Each agent explores a solution using the algorithm in Fig. 1. Although we give the number of evaluations to each agent as a constant, the number of repetitions is different among agents depending on the value of n given to each agent.

A-SAGAβ explores a solution using multiple agents, and evaluates exploration efficiency of those agents.

Exploration efficiency is evaluated as a cumulative sum of increased values in fitness of the elite individual [1]. We do not consider the increased value in the fitness when

[1] the individual with the best fitness value among all individuals.

1. receive individuals
2. evaluate individuals
3. select some individuals
4. crossover the selected individuals
5. mutate individuals
6. evaluate individuals
7. if cumulative evaluation times exceed a threshold, send to other agents some individuals as immigrants
8. otherwise, go to step 3.
9. return the resulting individuals

Figure 3. A-SAGA algorithm in each agent

1. initialize all agents
2. initialize all individuals
3. take over individuals to the next era agent
4. explore a solution by all agents
5. if it's not the last era, go to step 3.
6. select some agents
7. crossover the selected agents
8. mutate agents
9. if repetition times exceed a threshold, go to step 2.

Figure 4. A-SAGA algorithm as the whole system

it is imported by immigrants. We regard the sum as the fitness of the agent, and search a good parameter vector using the meta-GA. We show the algorithm of A-SAGAβ in Fig. 2.

3.4 Preliminary experiments and discussion

We have carried out some experiments using A-SAGAβ algorithm. However, we could not get expected performance as in Sect. 4.1.

In GAs, parameter vectors with good exploration efficiency vary throughout exploration phase [7]. A-SAGAβ evaluates agents with good efficiency in the early stage of exploration better than in the latter stage. Therefore, in the near final stages of exploration phase, agents which have shown good efficiency in the early stages dominate over other agents. We thought that's a reason why A-SAGAβ could not achieve good performance.

3.5 The improved algorithm

According to the discussion in the previous section, we have adopted a technique to divide a exploration phase into several units called *era* so that different agents explore solutions in multiple eras, respectively. We call the improved algorithm as A-SAGA. In A-SAGA, we can adapt parameter vectors in smaller granularity throughout the exploration phase.

We use the following parameters in addition to the parameters used in the meta-GA.

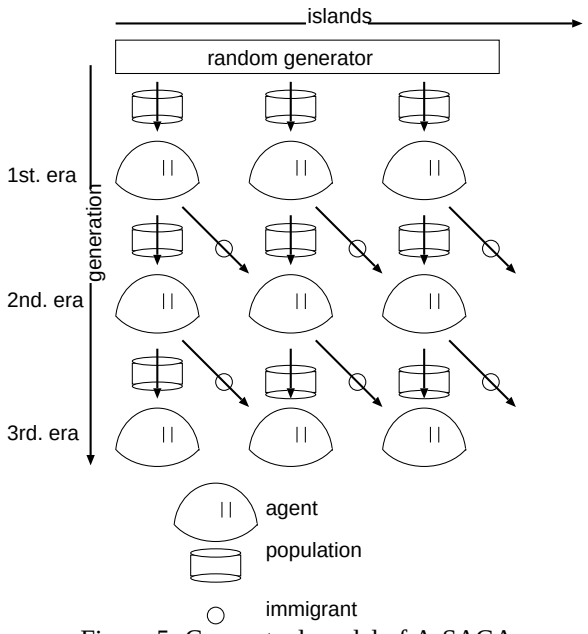

Figure 5. Conceptual model of A-SAGA

E_{m} : evaluation times per each meta-GA generation
G_{i} : the number of eras per each meta-GA generation
N_{a} : the number of agents per each era

For each agent in A-SAGA, parameter vector $\mathbf{v} = (e_{\mathrm{p}}, p_{\mathrm{m}}, p_{\mathrm{c}}, l)$ is given. Here, p_{m} and p_{c} denote the mutation rate and the crossover rate, respectively, l denotes the linear scheduling coefficient, and e_{p} denotes the number of evaluations per each generation of the GA within each agent.

In A-SAGA, agents in the first era initialize individuals, and the evolved individuals are taken over to the agents in the next era as shown in Fig. 5.

In A-SAGAβ, the number of individuals varies depending on parameters, while the maximum number of individuals are always given to agents in A-SAGA. Here, note that e_{p} individuals are evaluated in each GA generation, and that each agent evaluates individuals constant times denoted by $\frac{E_{\mathrm{m}}}{G_{\mathrm{i}} N_{\mathrm{a}}}$. We show the algorithm of each agent in Fig. 3 and the algorithm of the whole system in Fig. 4.

4 Experimental results

In order to evaluate efficiency of our algorithm, we have carried out some experiments on the Rastrigin function (equation 1) and Traveling Salesman Problem (TSP) called eil51[10]. Here, the number of Rastrigin function variables is ten, and each variable is expressed by 16 bits gray code. Both problems are minimization problems.

$$f(x_1, x_2, \ldots, x_n) = 10n + \sum_{i}^{n}(x_i^2 - 10\cos(2\pi x_i)) \quad (1)$$
$$-5.12 < x_i \leq 5.12$$

In order to measure improvement of exploration efficiency in A-SAGA, we have fixed the number of evaluations per meta-GA generation and measured the variation of the fitness value of the elite individual in each meta-GA generation (Experiment 1).

We have executed A-SAGA 2000 times until 20th meta-GA generation. In this experiment, we have used the following configuration.

E_{m} : 20000
G_{i} : 1, 10, 20
N_{a} : 10

Note that A-SAGA is equal to A-SAGAβ when $G_{\mathrm{i}} = 1$.

In A-SAGA, there are G_i eras in each meta-GA generation and individuals initially assigned to an agent are taken over to the agents in the succeeding eras. So, the parameter vector calculated in an agent in the first era also changes G_i times as era changes. According to the above discussion, in the last (20th) meta-GA generation, we have measured the value of each agent's parameter vector in each era to see how well the parameter vector is adapted (Experiment 2).

4.1 Result

In Fig. 6 and Fig. 7, the improvements of exploration efficiency by parameter adaptation when we applied A-SAGA to Rastrigin function and TSP eil51 are shown, respectively (Experiment 1). In the figures, meta-GA generations are represented by the horizontal axis, and the fitness value of the elite individual is represented by the vertical axis. In Fig. 6, note that the lines of $G_{\mathrm{i}} = 10$ and $G_{\mathrm{i}} = 20$ are overlapped.

In Fig. 8 and Fig. 9, the variations of average parameter values of p_{m} and e_{p} in the 20th meta-GA generation are shown, respectively, when we applied A-SAGA to the Rastrigin function (Experiment 2). Also, in Fig. 10 and Fig. 11, the results when we applied the same experiments to TSP eil51 are shown. In these figures, the progress of eras in the 20th meta-GA generation is represented by the horizontal axis, and the value of each parameter is represented by the vertical axis. Since we use the average value of all individuals in each era, it is obvious that the parameter value does not change when $G_{\mathrm{i}} = 1$.

4.2 Discussion

In Fig. 6, in all eras,it is shown that exploration efficiency of later generations is higher than earlier generations. Also, exploration efficiency is higher if G_{i} is 10 or 20. In Fig. 7, it is also shown that exploration efficiency is higher if G_{i} is 10 or 20 though differences are not as much as that

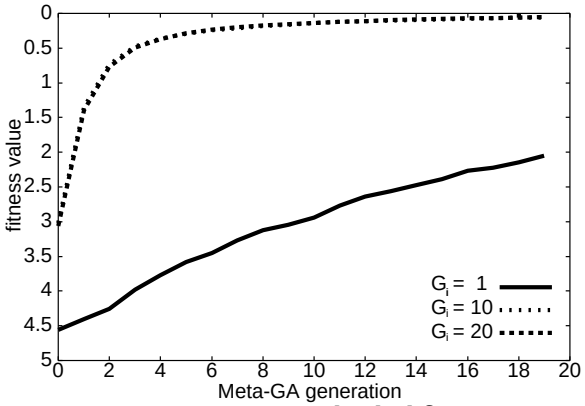

Figure 6. Rastrigin individual fitness

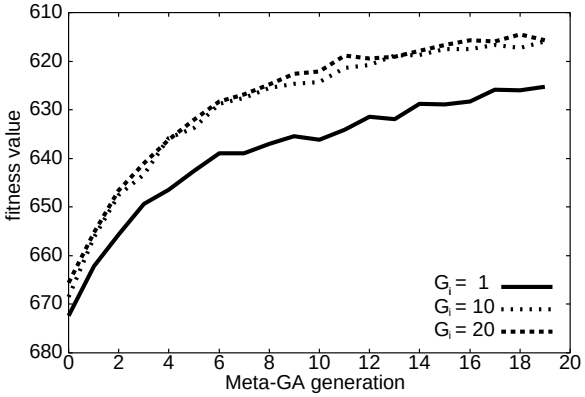

Figure 7. TSP eil51 individual fitness

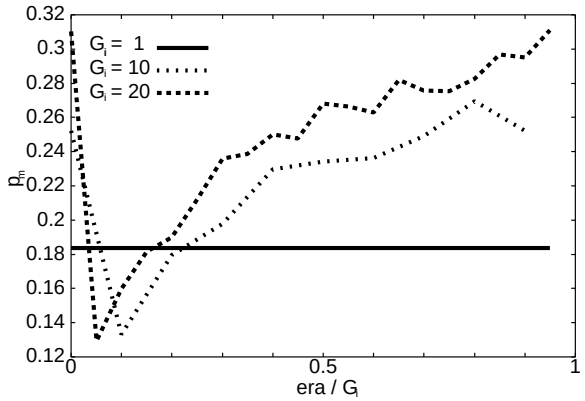

Figure 8. Rastrigin p_{m}

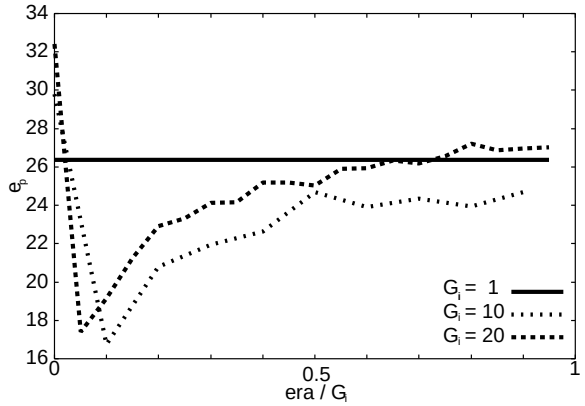

Figure 9. Rastrigin e_{p}

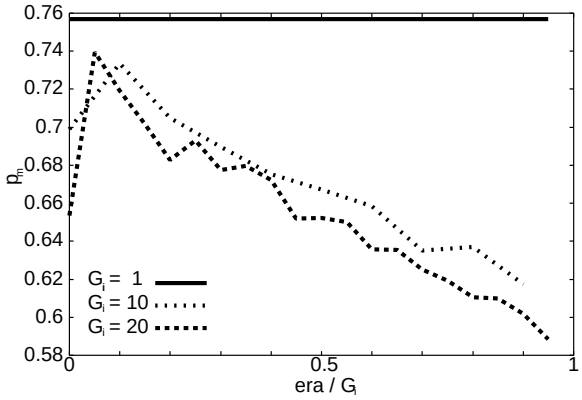

Figure 10. TSP eil51 p_{m}

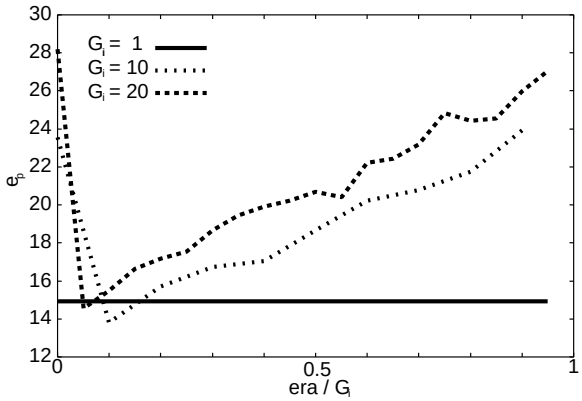

Figure 11. TSP eil51 e_{p}

in Fig. 6. These results show that the technique which divides a exploration phase into several units has improved exploration efficiency.

In Fig. 6 and Fig. 7, it is shown that exploration efficiency when G_{i} is 10 or 20 is higher than that when G_{i} is 1 even in the beginning of the first meta-GA generation. We consider that the agents which happen to be assigned good parameter values exist in higher probability when the number of agents ($G_{\mathrm{i}} \times N_{\mathrm{a}}$) is large. This is one of merits of A-SAGA owing to multi agents techniques similar to GAs with distributed environment scheme.

In Fig. 8, it is shown that the mutation rate increases as eras progress when G_{i} is 10 or 20. In general, in the near last phase of exploration, a high mutation rate is required to escape from the local minima. Existing adaptive GAs have low mutation rate in the last phase of exploration [4]. However, A-SAGA well adapts the mutation rate even in the last phase of exploration as shown in Fig. 8.

In Fig. 10, it is shown that the mutation rate decreases as eras progress. In TSP eil51, we used swapping the traveling order of any two cities as mutation operator. For this kind of problems, mutations hardly contribute to improvements of fitness values in the last phase of exploration. So, it is shown that A-SAGA well adapts parameters even in such a case.

In Fig. 9 and 11, the same tendency is shown, where a large value is initially given as e_{p}, then the value decreases quickly, and after that it increases as eras progress. In the beginning of the first era, A-SAGA needs rather large population of individuals since it needs diversity of population to find seeds of good solutions. Once agents find good seeds, it will be efficient to crossover small number of individuals close to the elite individual until getting to local minima. That's why the value of e_{p} decreases quickly. However, as eras progress, more diversity is needed to escape from local minima. That's why the value of e_{p} increases gradually.

In the meta-GA, sometimes, long exploration with bad parameters can find good solution rather than short exploration with good parameters. It will be similar in A-SAGA. The value of E_{m} must be decided by users.

For example, in Fig. 7, A-SAGA gets fitness value 615 for the average by $E_{\mathrm{m}} \times$ the number of meta-GA generations = $20000 \times 20 = 400000$ evaluations. In another experiment where $E_{\mathrm{m}} = 40000$ for one meta-GA generation, A-SAGA got the fitness value of approximately 600 for the average.

We can develop the algorithm which monitors exploration phase, and automatically decides E_{m} value.

5 Conclusion

We proposed a new adaptive GA called A-SAGA based on multi agent techniques which combines the existing meta-GA techniques and GA with distributed environment scheme. A-SAGA could simultaneously adapt for four parameters during exploration process in reasonable computation costs. In the future work, we would like to compare A-SAGA with meta-GA or other GAs in experiment. Moreover, we would like to allow agents to use other algorithm and to observe behavior of other agents.

References

[1] T. Bäck, "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] E. Cant'u-Paz, "A Survey of Parallel Genetic Algorithms", Calculateurs Pralleles, Reseaux et Systems Repartis, Paris: Hermes, 10(2):pp. 141–171, 1998.

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

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

[5] D. Goldberg, “Genetic Algorithm in Search, Optimization, and Machine Learning”, Addison-Wesley, Reading, MA, 1989.

[6] J. E. Hopcroft, J. D. Ullman, “Introduction to Automata Theory, Languages, and Computation”, Addison-wesley Publishing Company, Inc., Chapter 13, 1979.

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

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

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

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