# *Chemical Genetic Algorithms* - Coevolutionary Genotype-Phenotype Mapping by Modeling of Metabolism in Cell

**Hidefumi Sawai**
Communications Research Laboratory
588-2, Iwaoka, Nishi-ku
Kobe, 651-2492 Japan
sawai@crl.go.jp

**Hideaki Suzuki**
ATR Human Information Science Laboratory
2-2, Hikari-dai, Keihanna Science City
Kyoto 619-0288 Japan
hsuzuki@atr.co.jp

**Abstract- A chemical genetic algorithm (CGA) in which several types of molecules (information units) react with each other in a cell is proposed. Translation from codons (short substrings of bits) in DNA to amino acids (real value units) is specified by a particular set of translation molecules created by the reaction between tRNA units and amino acid units. This adaptively changes and optimizes the fundamental genotype-phenotype mapping during evolution. Through the struggle between cells containing a DNA unit and small molecular units, the codes in DNA and the translation table described by the small molecular units coevolve, and a specific output function (protein), which is used to evaluate a cell's fitness, is optimized. To demonstrate the effectiveness of the CGA, the algorithm is applied to a set of deceptive problems and the Shekel's foxholes problem with epistasis, and the results by using the CGA are compared to those by using a simple GA. It is shown that the CGA becomes a powerful strategy for optimizing functions that are hard to solve with the conventional GA.**

## 1 Introduction

An evolutionary system's ability to evolve a variety of adaptive functions or solutions (evolvability) is specified by a set of fundamental functional units. In a real biological system, the set comprises twenty amino acids, and genetic information written in DNA is translated into these units by using a set of translation molecules known as *aminoacyl-tRNAs* [Alberts *et al.*1994]. The aminoacyl-tRNAs define the fundamental mapping from genotype to phenotype, a fitness landscape on the DNA genotype space, and are one of the key molecules determining a biological system's ability. At an early stage of biological evolution, life succeeded in choosing an appropriate set of aminoacyl-tRNAs (translation relation from codes to amino acids) [Bedian2001, Wills2001], which enabled life to evolve a variety of adaptive functions or higher organisms like dinosaurs or mammals. This poses a fundamental question: how was the appropriate set of translation molecules chosen during biological evolution?

A recent study by one of the authors [Suzuki2000a, Suzuki2000b, Suzuki2001] attempted to answer the above question. His basic idea is that an artificial system's evolvability is enhanced by an objective measure. He proposed a measure for evolvability as well as the approach of numerically optimizing fundamental functional units prior to an experimental evolution run. In biological systems, however, the fundamental functional units (amino acids) and the fundamental code translation (aminoacyl-tRNAs) did not evolve prior to the evolution of codes (RNA or DNA). It is natural to assume that a number of different translation relations had been variously chosen and tested while assessing and evolving DNA codes, and in this sense the codes and the code translation simultaneously evolved (coevolved). When we design an artificial evolutionary system, the introduction of *coevolution* between codes and code-translation might help explore a better translation relation than a man-made relation and improve the evolutionary performance.

Based on the above notion, this paper introduces the coevolution between genetic codes and the translation table into genetic algorithms (GAs). Though GAs were originally invented through an inspiration to mimic the evolutionary strategies of living things [Holland1992], since Goldberg [Goldberg1989a] established a simple form of GA (simple GA, or SGA) and proved that it could be successfully applied to a real engineering problem, many GA studies have followed this line: a population of chromosomes (DNA strings) are prepared and subjected to genetic operations such as selection, mutation, and crossover. However, a selection unit in biological evolution is basically not a chromosome but a cell. A cell not only includes DNA strands for genetic codes but also functional units (amino acids) and translation

0-7803-7804-0/03/$17.00 © 2003 IEEE

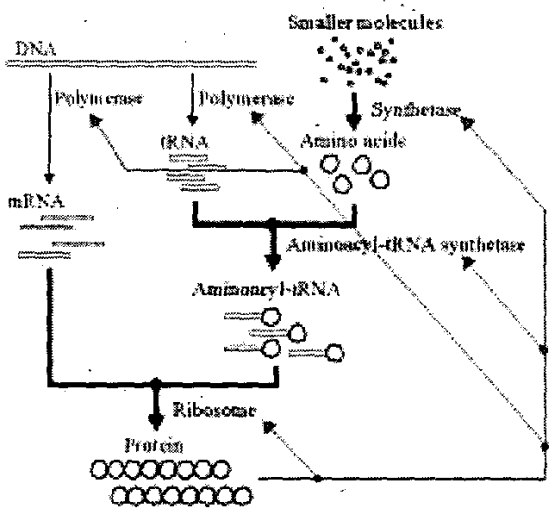


Figure 1: Biochemical reactions for the translation of genetic information in a living cell. The rectangles represent informational molecules such as DNA and RNA, and the large circles represent functional molecules (amino acids).

molecules (aminoacyl-tRNAs). Genetic information on DNA has no meaning without the construction of these smaller components in a cell [Bedian2001, Wills2001]. Imitating this biological structure, we prepare an artificial cell that includes a DNA string, a set of aminoacyl-tRNAs, a set of tRNAs, and a set of indexed amino acids. The tRNAs and amino acids are respectively created by the transcription and translation of the DNA, and the aminoacyl-tRNAs are created by the chemical reaction between tRNAs and indexed amino acids. A population of cells having this structure is evolved by using operations for selection, DNA mutation, DNA crossover, molecular exchange, and chemical reaction. The cell's fitness is evaluated from the target function, which is calculated from the specific output amino acid values (Fig. 1). To assess the validity of the proposed algorithm (which we refer to as the *chemical genetic algorithm* or CGA), we apply it to a few functional optimization problems that are hard to solve by SGA. Numerical experiments show great advantage of CGA vs. SGA for the tested optimization problems.

## 2 Modeling of Metabolism in Cell

A major lesson from the biological translation system (Fig. 1) is the changeability of the mapping between a genotype and a phenotype. In life, the basic mapping from a codon (genotype unit) to an amino acid (phenotype unit) is specified by a set of translation molecules, aminoacyl-tRNAs. The aminoacyl-tRNAs are created in reference to the information on tRNA derived from DNA; consequently, the translation table can be changed by modifying the genetic information in the DNA.

Another important point in life is the interdependence between molecules. The information for the production of all molecules in a cell comes from the DNA strands, and at the same time, the evolution of the information on DNA is influenced by the smaller molecules because the selective advantage of the DNA is determined by proteins created by the translation. In this sense, the codes (DNA information) and the code translation (repertory of aminoacyl-tRNAs) coevolve, which optimizes the system evolvability during evolution.

In our model that imitates the translation scheme of a biological system, we prepared four different types of molecular units in a cell for CGA: a DNA string, aminoacyl-tRNA units (aa-tRNAs), tRNA strings, and indexed amino acid units (iAminos) (Fig. 2). A DNA unit and the tRNAs are represented by binary strings, and the aa-tRNAs and iAminos are represented by combinations of a binary string and a real value. All of the proteins catalyzing the reactions are omitted. The parameters used in the model are summarized below.

- $J_1$ : Number of bits in a codon.
- $J_2$ : Number of bits in an index.
- $K$ : Number of output amino acids for the target protein, or the number of dimensions for the target function $F$.
- $L$ : Number of tRNA units in the DNA.
- $M$ : Number of codons for iAminos in the DNA.
- $I$ : Total length of (number of bits in) the DNA $= KJ_1 + L(J_1 + J_2) + MJ_2$.
- $L_1$ : Maximum number of tRNAs in a cell.
- $M_1$ : Maximum number of iAminos in a cell.
- $R_1$ : Initial/maximum number of aa-tRNAs in a cell.
- $R_2$ : Number of new aa-tRNAs created by the reaction between tRNAs and iAminos per cell per generation.
- $P_m$ : Mutation rate (probability of flipping of DNA bits) per bit per generation.
- $P_c$ : Crossover rate (occurrence probability of one-point crossover between a DNA pair) per cell pair per generation.
- $N$ : Population size (the number of cells in

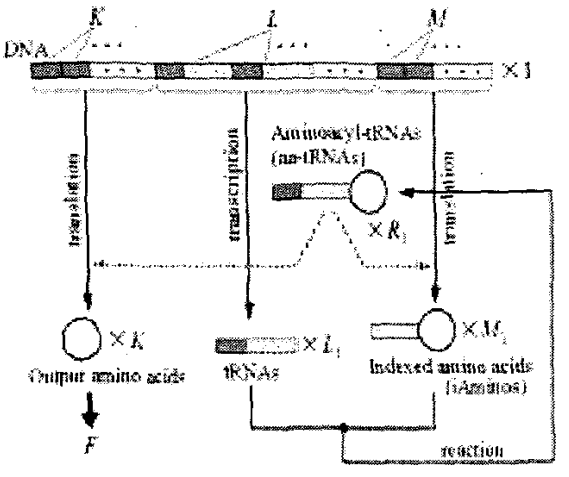


Figure 2: A cell structure used in the CGA. Dark hatched rectangles are codons described with $J_1$-bit strings, bright hatched rectangles are indexes described with $J_2$-bit strings, and circles are amino acids described with real numbers.

the population),
$\beta$ : Exponent for the target function $F$.
$a,b$ : Fitness coefficient for linear scaling.
$c$ : Fitness coefficient for exponential scaling.

The reactions in Fig. 2 proceed as follows. Every generation, $L$ tRNAs are created by the transcription of a DNA string, and $M$ iAminos are created by the translation of DNA. The *indices* for the iAminos are the implementation of the amino acid identifiers on the real tRNA units in a biological cell. Though there is no 'index' data for a biological amino acid, we explicitly attach an index string to every iAmino to enable matching between a tRNA and an amino acid. The newly created tRNAs and iAminos are mixed with older tRNAs and iAminos, and if the total molecule numbers exceed $L_1$ and $M_1$, respectively, some of the molecules are randomly chosen and eliminated. Then, the reaction between tRNAs and iAminos is put into action. A pair of randomly chosen tRNA and iAmino is compared and, if their indices are the same, a new aa-tRNA is created. The aa-tRNA's *codon* is copied from the tRNA's codon, its index is copied from the tRNA's or the iAmino's index, and its amino acid is copied from the iAmino. This process is repeated until $R_2$ aa-tRNAs are created, and then some of the aa-tRNAs are randomly chosen from the new or older aa-tRNAs and eliminated so that the total number of aa-tRNAs does not exceed $R_1$.

A set of aa-tRNAs created in this way is used for the codon-amino acid translation. To translate a codon ($J_1$-bit string) in the DNA string, an aa-tRNA is randomly chosen from the aa-tRNA set, and if its codon is the same as the codon on the DNA, its index and amino acid value are copied to create a new iAmino or a new output amino acid. Every generation, $K$ output amino acids are calculated in this way and used to evaluate the fitness value of the cell. In the selection operation (see below), every cell is assessed with the fitness value of the $K$-amino protein. This model can be regarded as a simplification of the selective environment wherein every cell is evaluated in terms of the efficiency of a single protein.

The entire procedure of the CGA is described as follows:

1. **[Initilization]** Prepare a population of $N$ cells with the archtecture shown in Fig. 2. In the initial state, no cell has an aa-tRNA, a tRNA, or an output amino acid; each cell only includes a DNA string and $M_1$ iAminos. The sequence of bits in the DNA string is randomly chosen for each strand. The iAminos (pairs of index and amino value) are also randomly chosen for each iAmino, but they are assumed to be common for all cells of the initial population.
2. **[Chemical Reaction]** Conduct the transciption, translation, and reaction described above for each cell.
3. **[Selection]** Calculate the fitness value from the output amino acids for each cell and conduct roulette-wheel selection using the fitness values. When a cell is reproduced, the entire information (all four kinds of molecules) is copied from the mother cell to the daughter cell.
4. **[DNA Mutation]** Conduct the conventional mutation (bit flipping) operation on the DNA strings of the cells.
5. **[DNA Crossover & Molecular Exchange]** Mate all cells to make $N/2$ pairs. For each pair, conduct the conventional crossover (exchange of DNA substrings) operation and a molecular exchange operation. In the latter operation, half of the aa-tRNAs, tRNA, and iAminos are randomly chosen for each parent cell and they are exchanged.
6. Examine the population, and terminate if a particular condition is satisfied. Otherwise, go to Step 2.

At the outset, every cell has $M_1$ different (but common for all cells) iAminos, so if $M_1$ is large enough, the amino acid diversity stored in the iAminos is sufficiently large. As evolution goes on, this diversity gradually decreases, and at the same time, the amino acid diversity in the aa-tRNAs increases. If by chance an appropriate aa-tRNAs is created by the reaction in a cell, the cell gets a higher fitness value than the others and its genetic information on the DNA strand and smaller molecules begins to spread not just through reproduction in the selection operation but also through the molecular exchange operations. An appropriate set of amino values is chosen in this way from a large initial amino acid repertory.

## 3 Experiments

### 3.1 Deceptive Problems

To verify the effectiveness of the CGA, we solve a functional optimization problem using the CGA. The problem includes three types of deceptive functions, types I to III, and is difficult to solve by conventional GAs such as a simple GA.

The fitness function is defined as $fitness = a + bF(x)$ for linear scaling or $fitness = \exp(cF(x))$ for exponential scaling where $a$, $b$ and $c$ are constant values. $F(x)$ is defined as:

$$F(x) = \left(\frac{1}{K}\sum_{k=0}^{K-1} f_k(x)\right)^{\beta},$$

where $\beta$ is a non-linearity factor. The deceptive function $F(x)$ is classified into the following three types, depending on the location of a global optimum in $K$-dimensional space.

Type I is a simple deceptive problem:

$$f_k(x) = \begin{cases} \alpha - x & \text{if } 0 \le x \le 0.8 \\ \frac{x-\alpha}{1-\alpha} & \text{if } 0.8 < x \le 1 \end{cases}$$

where $\alpha = 0.8$.

Type II is a medium-complex deceptive problem:

$$f_k(x) = \begin{cases} \alpha - x & \text{if } 0 \le x \le 0.8 \\ \frac{x-\alpha}{1-\alpha} & \text{if } 0.8 < x \le 1 \end{cases}$$

or

$$f_k(x) = \begin{cases} \frac{1-\alpha-x}{1-\alpha} & \text{if } 0 \le x < 0.2 \\ x - 1 + \alpha & \text{if } 0.2 \le x \le 1 \end{cases}$$

is randomly chosen according to each dimension $k$ ($k = 0, 1, \cdots, K-1$), where $\alpha = 0.8$.

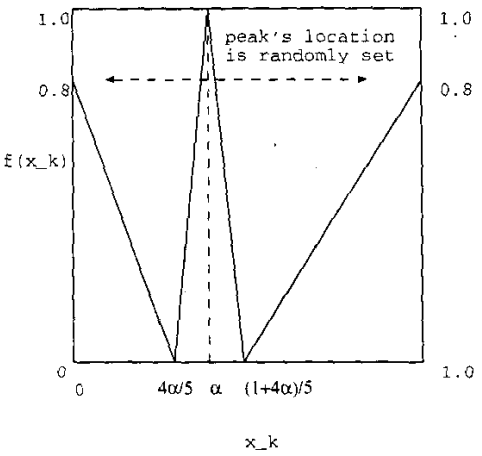


Figure 3: Type III: complex deceptive problem. Type I and II deceptive problems are in some sense special cases of type III where the peak's location is fixed at the point $x_k$=1.0, and randomly set at either $x_k$=0.0 or $x_k$=1.0, respectively.

Type III is a complex deceptive problem:

$$f_k(x) = \begin{cases} -\frac{x}{\alpha_k} + \frac{4}{5} & \text{if } 0 \le x < \frac{4}{5}\alpha_k \\ \frac{5x}{\alpha_k} - 4 & \text{if } \frac{4}{5}\alpha_k \le x \le \alpha_k \\ \frac{5(x-\alpha_k)}{\alpha_k - 1} + 1 & \text{if } \alpha_k \le x \le \frac{1+4\alpha_k}{5} \\ \frac{x-1}{1-\alpha_k} + \frac{4}{5} & \text{if } \frac{1+4\alpha_k}{5} < x \le 1. \end{cases}$$

where $\alpha_k$ is a different random number between 0 and 1 depending on each dimension $k$ ($k = 0, 1, \cdots, K-1$).

Type I is a simple deceptive problem where the global optimum is located at $x_k = 1.0$ ($k = 0, \ldots, K-1$). The $2^K - 1$ local optima exist at locations with either $x_k = 0$ ($k = 0, \cdots, K-1$). Type II is a medium-complex deceptive problem where the global optimum is located at either $x_k = 0$ or 1 ($k = 0, \cdots, K-1$) and is randomly chosen according to each dimension. Similar to Type I, the number of local optima is $2^K - 1$. Type III is a complex deceptive problem where the location of the global optimum is randomly set in $K$-dimensional space; however, unlike types I and II, the number of local optima is $3^K - 1$ because the locations at both $x_k = 0$ and 1 are local optima in each dimension.

The attractor with a global optimum only has a length of 0.2, whereas the attractors with local optima have much more and wider regions in $K$-dimensional space. The region with local optima is $5^K - 1$ times wider than the region with a global optimum for all types of deceptive problems.

Experiments were performed for the three types of deceptive problems in five- and ten- dimensional cases. According to some preliminary experiments, we set the following experiment conditions: $J_1$ is set as 4 and 6 for the CGA in five and ten dimensions, respectively. Fitness as exponential scaling for the SGA was not adequate because none of the ten runs was successful because exponential scaling was too strong for the SGA to succeed in finding a global optimum for all types of deceptive problems. On the other hand, linear scaling was also not adequate for the CGA because none of the runs was successful to find a global optimum for the complex deceptive problem. Therefore, linear scaling and exponential scaling of the fitness function was adopted for the SGA and the CGA, respectively. A DNA string in the SGA is the left part of the CGA string. Its length is $J_1 \times K$ bits, and the codons in the SGA are interpreted as real values by using the binary coding method. $P_m$ and $P_c$ were respectively set to 0.005 and 0.7 in common for both GAs, the number of cells (i.e., population size) $N$ was set to 256, and the roulette-wheel selection with an elite selection of one individual was adopted for the selection operation. The non-lineality factor $\beta$ in $F(x)$ was set to 5. Twenty different runs with different random seeds were conducted for the SGA and CGA in both five- and ten-dimensional cases. Table 1 shows the set of parameters used for the CGA.

Table 1: Parameter set for CGA

| $K$ | $J_1$ | $J_2$ | $L$ | $M$ | $L_1$ | $M_1$ | $R_1$ | $R_2$ | $\mathrm{I}$ |
|---|---|---|---|---|---|---|---|---|---|
| 5 | 4 | 8 | 16 | 32 | 80 | 1280 | 80 | 16 | 340 |
| 10 | 6 | 8 | 64 | 128 | 320 | 1280 | 320 | 64 | 1724 |

### 3.2 Shekel's Foxholes Problem

The second problem is Shekel's foxholes function. Although the original form of this function is the one whose domain is $0 \leq x_k \leq 10$ and whose range is negative, here we normalize it so that the domain might be $0 \leq x_k \leq 1$ and the range might be positive. Therefore, the functional value is given by

$$F(x) = \sum_{j=1}^{m} \frac{1}{\sum_{k=1}^{K} (10 \cdot x_k - a_{jk})^2 + c_j},$$

where $m = 30$ and $a_{jk}$ and $c_j$ are constant numbers fixed in advance. From preliminary experiments, we observed that for this function, the CGA's performance is much influenced by the scaling method, whereas the SGA's performance is not so influenced by the scaling. In the following, we take the exponential scaling with $c = 50$ for the CGA as the best condition obtained from the preliminary experiments, and take the linear scaling with $a = 0$, $b = 1$ for the SGA. The other parameters are the same as those used in the previous subsection: $P_m = 0.005$ (mutation rate), $P_c = 0.7$ (crossover rate), and $N = 256$ (population size). Using different random number sequences, we conducted ten different runs with the SGA and CGA for the five-dimensional ($K = 5$) foxholes problem.

## 4 Results

### 4.1 Deceptive Problems

Table 2: Success ratio in SGA and CGA

| GA | SGA | SGA* | CGA | SGA | SGA* | CGA |
|---|---|---|---|---|---|---|
| $K$ | 5 | 5 | 5 | 10 | 10 | 10 |
| $J_1$ | 4 | 4 | 4 | 6 | 6 | 6 |
| scaling | linear | linear | exp. | linear | linear | exp. |
| Type I | 10/20 (50%) | 15/20 (75%) | 20/20 (100%) | 0/20 (0%) | 2/20 (10%) | 20/20 (100%) |
| Type II | 8/20 (40%) | 11/20 (55%) | 20/20 (100%) | 0/20 (0%) | 2/20 (10%) | 20/20 (100%) |
| Type III | 1/20 (5%) | 1/20 (5%) | 20/20 (100%) | 5/20 (25%) | 19/20 (95%) | 6/20 (30%) |

For each condition, the number of successful runs out of twenty different trials is shown with the success ratio in parentheses. For example, 8/20 (40%) means that eight runs out of twenty runs succeeded in finding the global optimum, so the success ratio is 40%. The SGA with asterisk(*) ran 10,000 generations.

Table 2 compares results between the SGA and CGA for five and ten dimensions. Fitness was defined as linear scaling with $a = 0$, $b = 1$ for the SGA and as exponential scaling with $c = 20.79$ for the CGA. Computational cost for the SGA and CGA to run 1,000 generations is respectively a few minutes and 20-30 minutes using a general-purpose SUN workstation. Therefore, the computational cost of the CGA is about ten times larger than that of the SGA. In Table 2, the SGA with asterisk(*) ran 10,000 generations for comparison to the CGA. Comparing the results between the SGA and CGA, almost all runs of the CGA were successful for the three types of deceptive problems, except for type III in ten dimensions, whereas the SGA didn't show as good performance as the CGA did, especially for ten

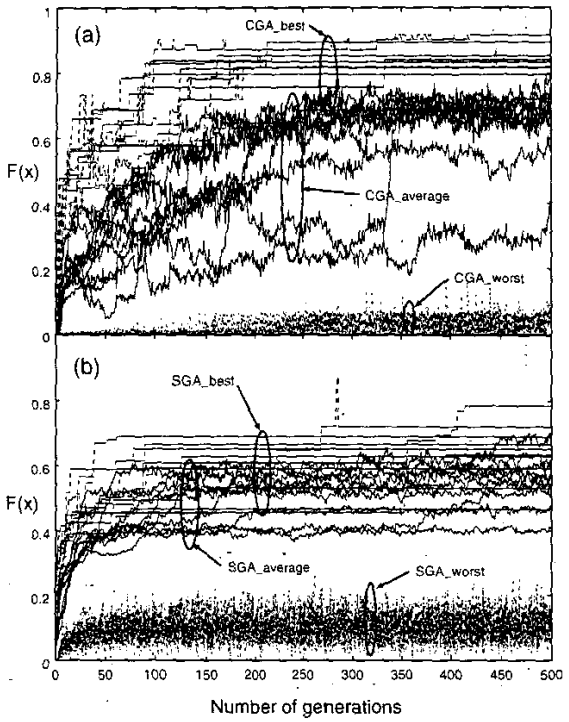


Figure 4: Evolution for the five-dimensional type III deceptive problem. (a) CGA with $J_1 = 4$ and (b) SGA with $J_1 = 10$. The solid fluctuating lines represent the average values, the doted lines represent the best values, and the lower fluctuating dashed lines represent the worst values. Ten different runs are superimposed.

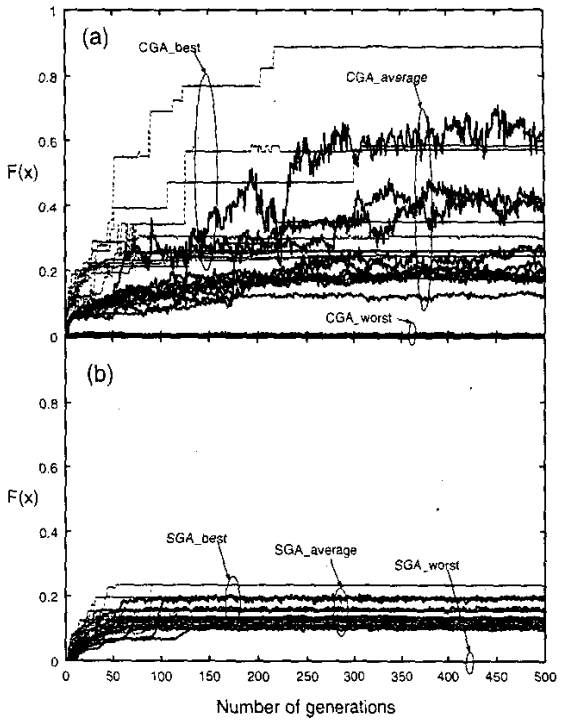


Figure 5: Evolution for the five-dimensional Shekel's foxholes problem. (a) CGA with $J_1 = 4$ and (b) SGA with $J_1 = 10$. Ten runs are superimposed. See the Fig. 4 caption for the description of lines.

dimensions.

Figure 4(a) shows the function values $F(x)$ of the CGA. Ten different runs are superimposed. Diversity during evolution of the CGA is well maintained because the best and average values are different from each other. Although the function values of the CGA rose up slowly, both the best and average values became better and better and finally reached a global optimum.

For comparison to the CGA, Fig. 4(b) shows the evolution of the SGA's function values for the complex deceptive problem in five dimensions. Almost all runs converged to local optima without converging to a global optimum because diversity in the function values was small due to the fact that the average and best values were approximately the same.

The number of different values in the iAminos was initialized as 1,280 and gradually decreased during evolution. On the other hand, the number of different values in the aa-tRNA was initialized as zero, suddenly increased at an early stage of generations, and then decreased as generations proceeded. Two kinds of numbers nearly converged to a final constant value, 22, at the 300th generation.

### 4.2 Shekel's Foxholes Problem

Figures 5(a) and (b) show the function values $F(x)$ for the CGA and SGA, respectively. Using ten trials of a run for five hundred generations, both CGA and SGA failed to find a global optimum solution which is judged by the criterion $F(x) > 0.9$. However, for the CGA, the $F(x)$ values are much fluctuated and improved during evolution, and the final $F(x)$ values achieved after five hundred generations range from 0.178 to 0.888 that is much close to the global optimum value. For the SGA, on the other hand, such improvement is not observed. After initial short improvement, the population enters into a long period of stasis during which the $F(x)$ values are kept constant. As a consequence, the the final

$F(x)$ values achieved by the SGA range from 0.121 to 0.235 which is much smaller than the global optimum value. Though the computational cost of the CGA is again ten times larger than that of the SGA, the CGA outperforms the SGA for the Shekel's foxholes problem, too.

## 5 Discussion

### 5.1 Comparing performance between CGA and SGA

To verify the performance of the CGA compared to that of the SGA, several kinds of problems were introduced and tested. According to the main results shown in Table 2, we can say that the CGA can find a global optimum solution far more often than can the SGA. This great advantage of the CGA over the SGA comes from the appropriate control of diversity in the population. As shown in Figs. 4(a) and 5(a), the CGA can recurrently generate a variety of amino acid values even after a population is stuck around the vicinity of a local optimum. Since the output amino values are affected by both the DNA sequence and the smaller molecules, the CGA can explore the search space more extensively than the SGA, which enables the CGA to escape from local optimums. This is especially the case for an early stage of evolution. For later stages of evolution, on the other hand, the CGA exhibits another behavior, the convergence to the optimum repertory of amino acids. As evolution goes on, the diversity in amino values gradually decreases. The coevolution between DNA and smaller molecules controls this convergence in an appropriate manner, so that the CGA can finally obtain the optimum translation relation. For the SGA, on the other hand, once a population is stuck at a local optimum, evolution enters a long period of stasis due to the lack of cell diversity as shown in Figs. 4(b) and 5(b). Because the translation relation from DNA to real output values is fixed through a run, it is very hard for the SGA to escape from a local optimum wherein a population of DNA strands loses the diversity at some loci. The SGA offers far smaller possibility of exploration than the CGA does.

### 5.2 Coding in GAs

Since their proposal, GAs have been studied from a variety of viewpoints: a number of different versions of genetic operations have been tested [Goldberg1989a], an island model and other population models have been proposed for maintaining genetic diversity in a population [Tanese1989, Mühlenbein1989], and various coding methods have been devised to solve engineering problems using binary string chromosomes [Caruana & Schaffer1988, Goldberg Korb & Deb1989b]. Among them, the coding is one of the most important factors in GAs because it determines the fitness landscape on the genotype space affecting the GA performance. Man-made codings such as binary, Gray [Caruana & Schaffer1988, Goldberg1989a], and other methods, [Goldberg Korb & Deb1989b, Wright1991, Kargupta1997] have been proposed to project a fitness function on the genotype space so that the GA operations might search for solutions more stably and more effectively. However, as discussed in Section 2, life did not determine the coding before its evolution. The coding, or in other words, the genotype-to-phenotype association, was coded in the DNA/RNA units and was changed and optimized together with the codes in DNA/RNA during evolution. The CGA introduced this mechanism to conventional GAs and achieved far better performance than the simple GA. This result suggests that further work is worthwhile for the evaluations and refinements of the CGA.

### 5.3 Comparing CGA to Other Co-evolutionary Methods

Co-evolutionary methods using such as co-evolution between a host and a parasite [Handa1997], symbiotic relations between co-evolving populations [Paredis1995] [Murao2002], and predator-prey relations [Hillis1991] have been proposed. The method [Handa1997] consists of two GAs, a host GA and a parasite GA. The host GA searches for the solutions, whereas the parasite GA searches for useful schemata in the host GA. The method [Paredis1995] is called SYMBIOT algorithm which uses two co-evolving populations. One population contains permutations (orderings), the other population consists of proposed solutions. In the method [Murao2002], the genotype-phenotype map is adaptively improved by exploring the map itself during the search process for solution candidates. The method [Hillis1991] shows an example of how simulated evolution via co-evolving parasites can be applied to sorting networks. The CGA is distinct from all other co-evolutionary methods mentioned above in terms of modeling of metabolism in a cell obtained through a very long period of evolutionary process.

### 5.4 Future problems

The present version of CGA is just a simple actualization of molecular reactions in a cell. The moleculer types prepared in Fig. 2 is a minimal set for the translation of DNA information, and some other important

molecules such as enzymes (proteins catalyzing reactions or synthesizing amino acids) are omitted. We consider introducing these molecules to CGA as an important problem for the future. Also, in the present model, the amino values are not synthesized but are initially created on indexed amino acids. Making amino values newly synthesized during evolution is also a future problem to be tackled. Applying the CGA to more functions with epistasis or real engineering problems and comparing its performance with more effective evolutionary algorithms other than the SGA are to be solved in the future.

## 6 Conclusions

We have developed a new bio-molecular algorithm, a chemical genetic algorithm (CGA), in which several types of molecules react with each other in a cell. Translation from codons in DNA to amino acids is specified by a particular set of translation molecules (aminoacyl-tRNAs), which are created by the reaction between tRNAs and amino acids. Those smaller molecules are created from DNA, so the codes in DNA and the code translation in smaller molecules coevolve in the model. During evolution, the fundamental genotype-phenotype mapping is adaptively changed and converges to the optimum one. Through the struggle between cells with a DNA strand and smaller molecules, a specific output function (protein), which is used to evaluate a cell's fitness, is optimized. To demonstrate the effectiveness of the CGA, the presented algorithm was applied to a set of deceptive problems and the Shekel's foxholes problem, and the results by using the CGA were compared to those by using a simple GA as a conventional GA. As a result, it was shown that in the constructed chemical reaction model, the coevolution between codes and code translation appropriately controls the diversity of population and makes the population to converge to the global optimum with far higher probability than the conventional SGA.

## Bibliography


[Alberts *et al.*1994] Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., Watson, J.D. 1994. *Molecular Biology of the Cell. The Third Edition.* New York: Garland Publishing.

[Bedian2001] Bedian, V. 2001. Self-description and the origin of the genetic code. *BioSystems* **60** 39-47

[Caruana & Schaffer1988] Caruana, R.A., Schaffer, J.D. 1988. Representation and hidden bias: Gray versus binary coding in genetic algorithms. In: Laird, J. (ed.): *Proceedings of 5th International Conference on Machine Learning.* San Mateo, CA: Morgan Kaufmann 153-161

[Goldberg1989a] Goldberg, D.E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning.* New York: Addison-Wesley

[Goldberg Korb & Deb1989b] Goldberg, D.E., Korb, B., Deb, K. 1989. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems* **3** 493-530

[Kargupta1997] Kargupta, H., Godlberg D.E. Wang, L. 1997. Extending the Class of Order-k Delineable Problems for the Gene Expression Messy Genetic Algorithm. *Proc. of the 2nd Annual Conf. on Genetic Programming* 364-369

[Holland1992] Holland, J.H. 1992. *Adaptation in Natural and Artificial Systems.* Boston: MIT Press

[Mühlenbein1989] Mühlenbein, H. 1989. Parallel genetic algorithms, population genetics and combinatorial optimization. *Proceedings of the 3rd International Conference on Genetic Algorithms.* 416-421

[Suzuki2000a] Suzuki, H. 2000a. Minimum Density of Functional Proteins to Make a System Evolvable. *Proc. of the 5th AROB in 2000.* Vol. 1, 30-33

[Suzuki2000b] Suzuki, H. 2000b. Evolvability Analysis: Distribution of Hyperblobs in a Variable-Length Protein Genotype Space. *Proc. of Artificial Life VII.* Cambridge: MIT Press, 206-215

[Suzuki2001] Suzuki, H. 2001. String Rewriting Grammar Optimized Using an Evolvability Measure. *Proc. of the 6th European Conference on Artificial Life.* Berlin: Springer-Verlag, 458-468

[Tanese1989] Tanese, R. 1989. Distributed genetic algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms.* 434-439

[Wills2001] Wills, P.R. 2001. Autocatalysis, information, and coding. *BioSystems* **60** 49-57

[Wright1991] Wright, A.H. 1991. Genetic algorithms for real parameter optimization. *Foundations of Genetic Algorithms (FOGA-1).* 205-218

[Hillis1991] Hillis, W.D. 1991. Co-Evolving Parasites Improve Simulated Evolutution as an Optimization Procedure. *Artificial Life II* vol.X, 313-324

[Paredis1995] Paredis, J. 1995. The Symbiotic Evolution of Solutions and their Representation. Proceedings of the sixth Int. Conf. on Genetic Algorithms. 359-365

[Handa1997] Handa, H., Baba, N., Katai, O., Sawaragi, T. and Horiuchi, T. 1997. Genetic Algorithm Involving Coevolution Mechanism to Search for Effective Genetic Information. *Proceedings of IEEE Int. Conf. on Evolutionary Computation* 709-714

[Murao2002] Murao, H., Tamaki, H. and Kitamura, S. 2002. A Coevolutionary Approach to Adapt the Genotype-Phenotype Map in Genetic Algorithm. *Proc. of Congress on Evolutionary Computation* 1612-1617