DYNAMIC BEHAVIOR CONTROL OF AUTONOMOUS MOBILE ROBOTS USING SCHEMA CO-EVOLUTIONARY ALGORITHM

Kwee-Bo Sim, Ho-Byung Chun, and Dong-Wook Lee

School of Electrical and Electronic Engineering, Chung-Ang University
221, Huksuk-Dong, Dongjak-Ku, Seoul 156-756, Korea
kbsim@cau.ac.kr

ABSTRACT

The theoretical foundations of Genetic Algorithms (GA) are the Schema Theorem and the Building Block Hypothesis. In the meaning of these foundational concepts, simple genetic algorithms (SGA) allocate more trials to the schemata whose average fitness remains above average. Although SGA does well in many applications as an optimization method, still it does not guarantee the convergence of a global optimum. Therefore as an alternative scheme, there is a growing interest in a co-evolutionary system, where two populations constantly interact and co-evolve in contrast with traditional single population evolutionary algorithms. In this paper, we propose n new design method of an optimal fuzzy logic controller using co-evolutionary concept. In general, it is very difficult to find optimal fuzzy rules by experience when the input and/or output variables are going to increase. So we propose a co-evolutionary method finding optimal fuzzy rules. Our algorithm is that after constructing two population groups made up of rule base and its schema, by co-evolving these two populations, we find optimal fuzzy logic controller. By applying the proposed method to a path planning problem of autonomous mobile robots when moving objects exist, we show the validity of the proposed method.

1. INTRODUCTION

The evolutionary algorithms (EAs) based on the natural selection theory have been studied widely as a solution of the intelligent information processing system. Typically genetic algorithm (GA) [1-3], genetic programming (GP) [4-5], evolutionary strategies (ES) [6-7], and evolutionary programming (EP) [8] belong to the categories of EAs, and these have been successfully applied to many different applications according to the data structure and genetic operators. The simple genetic algorithm (SGA) was proposed by J. H. Holland [1] as a computational model of living system's evolutionary process and has become popular as a population-based optimization method. Although SGA provides many opportunities to obtain a global optimal solution, the performance of SGA is more or less limited depending on the predefined fitness function given by a system designer. It is said, therefore, that SGA works on static fitness landscapes [5].

Natural evolution, however, works on dynamic fitness landscapes that change over evolutionary time as a result of co-evolution. Also it is believed that co-evolution between different species or different organs results in the current state of complex natural systems. There are many types of co-action between different species. The co-action between two different populations has been very important subject in ecology. In ecology the types of co-action are classified into positive (+) co-action and negative (-) co-action according to the result of co-action. From this point of view, there is a growing interest in co-evolutionary systems, where two populations constantly interact and co-evolve in contrast with traditional single population-based evolutionary algorithms. Also it is believed that these kinds of co-evolutionary methodology are more similar to biological evolution in nature.

Generally the co-evolutionary algorithms can be classified into two categories, which are predator-prey co-evolution [9-10] and symbiotic co-evolution [11]. Predator-prey relation is the most well known example of natural co-evolution. As future generations of predators develop better attacking strategies, there is a strong evolutionary pressure for preys to defend themselves better. In such arms races, success on one side is felt by the other side as failure to which one must respond in order to maintain one's chances of survival. This, in turn, calls for a reaction of the other side. This process of co-evolution can result in a stepwise increase in complexity of both predator and prey [9]. Hillis [10] proposed this concept with a problem of finding minimal sorting network for a given number of data. Also co-evolution between neural networks and training data was proposed in the concept of predator and prey [12]. Also a new fitness measure in a co-evolutionary algorithm has been discussed in terms of dynamic fitness landscape. Leigh van Valen, a biologist, has suggested that the "Red-Queen effect" arising from co-evolutionary arms races has been a prime source of evolutionary innovations and adaptations [13]. This means that the fitness of one species changes depending on the other one.

In this paper, we introduce schema co-evolutionary algorithm (SCEA) and an extended schema theorem from the SCEA, where the fitness of a population changes according to the evolutionary process of the other population. Also we present how the SCEA works including fitness measure. As a result of co-evolution the optimal solution can be found more reliably in a short time with a small population than SGA. We show why the SCEA works better than SGA in terms of an extended schema theorem and parasitizing process. And, we propose a co-evolution method generation optimal fuzzy rule base, where the fitness of a population changes according to the evolution process of the other population. We present how to extract fuzzy rules using the SCEA. In general, it is very difficult to find fuzzy rules by hand when the input-output variables are going to increase. In this paper, therefore, we extract fuzzy rules by co-evolving the fuzzy rules and their schema.

To show the effectiveness of the proposed method, we applied our method to autonomous mobile robotic system, the objective of which is finding a goal and avoiding static/moving obstacles.

2. SCHEMA CO-EVOLUTIONARY ALGORITHM

Like the other co-evolutionary algorithms, the SCEA has two different, still cooperatively working populations: a host-population and a parasite-population. The former is made up of the candidates of solution and works about the same as conventional genetic algorithm. The latter is a set of schemata, which is to find useful schemata called "Building Block" [2], [3]. Figure 1 shows an overview of the SCEA.

SGA has four major steps for one generation, which are evaluation, selection, crossover, and mutation. Our algorithm, however, has an additional step, parasitizing, before the selection. After all strings in the host-population are evaluated, some of them are selected randomly for each schema of the parasite-population and then parasitized by the corresponding schema. We evaluate the strings newly generated from parasitizing and then measure the fitness improvement between the original string and the parasitized one. We replace the parasitized string having the largest improvement value with the corresponding string for each schema. Using the amount of the improvement we can assign the fitness of each schema in the parasite-population. Therefore the fitness of each schema in the parasite-population indicates the usefulness of the schema. We apply the same process of the SGA to the parasite-population after fitness assignment. Now we explain the parasitizing process in detail.

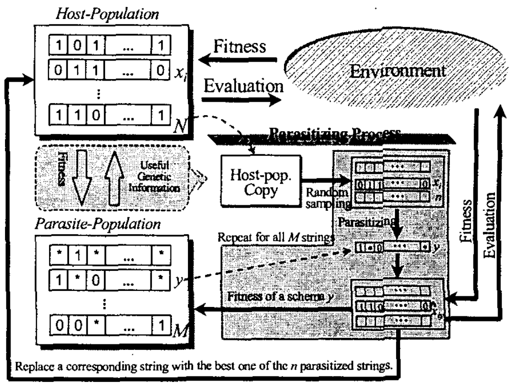


Fig. 1. A block diagram of the SCEA

SGA has four major steps for one generation, which are evaluation, selection, crossover, and mutation. Our algorithm, however, has an additional step, parasitizing, before the selection. After all strings in the host-population are evaluated, some of them are selected randomly for each schema of the parasite-population and then parasitized by the corresponding schema. We evaluate the strings newly generated from parasitizing and then measure the fitness improvement between the original string and the parasitized one. We replace the parasitized string having the largest improvement value with the corresponding string for each schema. Using the amount of the improvement we can assign the fitness of each schema in the parasite-population. Therefore the fitness of each schema in the parasite-population indicates the usefulness of the schema. We apply the same process of the SGA to the parasite-population after fitness assignment. Now we explain the parasitizing process in detail.

As above-mentioned, the parasite-population searches useful schemata and delivers the genetic information to the host-population by parasitizing process. We explain this parasitizing process with the fitness measure of the parasite-population and the alteration of a string in the host-population. Figure 2 shows the parasitizing process. The fitness of a schema in the parasite-population depends on the n strings sampled in the host-population. In the context of a computational model of co-evolution, the parasitizing means that the characters of a string are replaced by the fixed characters of a schema. The other

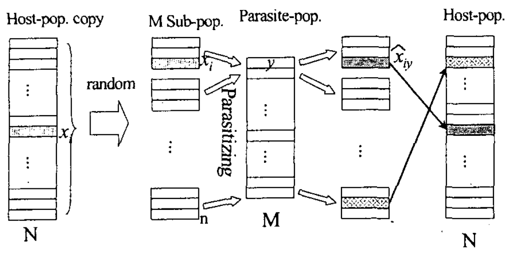


Fig. 2 Parasitizing process (Replace a string with the best one of the n parasitized strings). N is the population size of the host-population, M is that of the parasite-population, and n is the size of each M sub-populations.

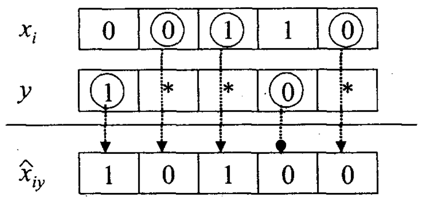


Fig. 3 An example of the parasitizing (x_i is parasitized by y into $\hat{x}_{iy}$).

positions of the string, i.e., the same positions of don't-care symbol (*) in the schema, hold their own values. Thus

$$\hat{x}_i^p = \begin{cases} y^p, & \text{if } p_{th} \text{ character of } y \text{ is fixed} \\ x_i^p, & \text{otherwise} \end{cases} \quad ,(0 \le p \le l-1) \quad (1)$$

where p is the index of the locus of a string, x_i^p is a value of p_{th} locus of a string x_i and l is the length of a string.

Figure 3 illustrates an example of the parasitizing. The process of the SCEA is, in brief, that a useful schema found by the parasite-population is delivered to the host-population according to the fitness proportionate, and the evolutionary direction of the parasite-population is determined by the host-population.

The fitness F_y of a string y in the parasite-population is determined as follows:

Step 1: Copy a host-population and then determine a set of strings to be parasitized, that is, select randomly n strings in the host-population copy which are parasitized by a schema y. Even though the same string is simultaneously selected for the other schema, it still has opportunities to explore different schema space depending on each schema.

Step 2: Let $x_1, \cdots, x_n$ be the sampled strings, and $\hat{x}_{1y}, \cdots, \hat{x}_{ny}$ be the parasitized strings. A parasiteized string is the sampled string that is parasitezed by a schema y.

Step 3: To determine the fitness of a string y in the parasite-population, we set a fitness function of one time parasitizing as the difference in the fitness.

$$\hat{f}_{iy} = f(\hat{x}_{iy}) - f(x_i), \qquad (i = 1, \cdots, n) \qquad (2)$$

where $f(x_i)$ is the fitness of a string x_i, and $f(\hat{x}_{iy})$ is the fitness of a parasitized-string $\hat{x}_{iy}$.

Step 4: For each of M schemata, the best individual, which has the largest improved value, is replaced by the corresponding one of the original host-population. When there are no improved ones in the sub-population there is no replacement (as shown second sub-population in Fig. 2). As a result of the replacement, the instances of useful (useless) schemata are increased (decreased) in the host-population. Furthermore new schemata which do not exist in the original host-population can be added.

Step 5: Since the parasite-population plays a role of finding useful schemata, the fitness F_y of a schema y is defined as the sum of the fitness improvement:

$$F_y \triangleq \sum_{i=1}^{n} \max[0, \ \hat{f}_{iy}] \qquad (3)$$

Equation (3) means that the fitness of a schema in the parasite-population is depending on the parasitized strings in the host-population. In the next sub-section, we derive an extended schema theorem from this schema co-evolutionary algorithm and show that it covers the GA-hard problems.

3. EXTENDED SCHEMA THEOREM

The SCEA is based on the Schema Theorem and the Building Block Hypothesis [2], [3]. First we discuss the original theoretical foundations of the genetic algorithm. SGA uses a population of genotypes composed of fixed-length binary strings called chromosome. SGA evaluates a population of genotypes with respect to a particular environment. The environment includes a fitness function that rates the genotype's viability. SGA reproduces genotypes proportionally to their relative fitness using a variety of genetic operators. One operator, termed crossover, uses the recombination of two parents to construct novel genotypes. The mutation operator creates new genotypes from a single parent with a probabilistic alteration.

The theoretical foundations of genetic algorithms rely on a binary string representation of solutions, and a notion of a schema. A schema is a subset of the search space, which matches it on all positions other than don't care symbol (*). There are two important schema properties, order and defining length. The number of 0 and 1 positions, i.e., fixed positions is called the order of a schema H (denoted by o(H)). And the defining length of a schema H is the distance between the first and the last fixed string positions (denoted by δ(H)). For example, the order of ***00**1** is 3, and its defining length is 4. An instance of a schema H is a bit string which has exactly the same bit values in the same positions that are fixed bits in H. For example, 1000, 1010, 1100, and 1110 are instances of a schema 1**0.

Another property of a schema is its fitness at generation k, denoted by $f(H,k)$. It is defined as the average fitness of all strings in the population matched by that schema H. Therefore, the combined effect of selection, crossover, and mutation on the expected number of a schema is formulated by:

$$m(H,k+1) \geq \frac{f(H,k)}{\overline{f}(k)} \cdot m(H,k) \cdot \left[1 - p_c \cdot \frac{\delta(H)}{(l-1)} - p_m \cdot o(H)\right] \quad (4)$$

where $m(H,k)$ is the number of instances of a schema H at generation k, $\overline{f}(k)$ is the average fitness of all individuals in the population, l is the number of bits in a string, p_c is the crossover rate, and p_m is the mutation probability. The above equation is known as the Schema Theorem [1]-[3] and means that the short, low-order, and above-average schemata, called as the Building Blocks, would receive an exponentially increasing number of strings in the next generations. If there does not exist a solution in the Building Blocks, however, simple genetic algorithm might fail to find that solution. The deceptive function is most well known as a problem violating above theorem. T. Kuo and S. Y. Hwang [9] showed that disruptive selection works better that directional selection on the deceptive functions.

Now we derive an extended schema theorem relevant to the SCEA, and show that it covers the deceptive functions. If a string y in the parasite-population represents a schema H, it is clear that the above parasitizing process can be interpreted, in the context of useful (useless) schemata, as a process of increasing (decreasing) the number of instances of a schema H in the host-population. If we recall the original schema theorem, the number of instances of a schema H at the generation k is changed by the amount of newly generated instances of that schema. As for the SCEA, the number of instances $m'(H,k)$ of a schema H in the host-population is formulated by

$$m'(H,k) = m(H,k) + \hat{m}(H,k) \quad (5)$$

where $m(H,k)$ is the original number of instances of a schema H in the host-population, and $\hat{m}(H,k)$ is the number of instances which are increased or decreased as a result of the parasitizing process.

Since the number of instances of a schema is increased when at least one of the parasitized strings has improved, it can be formulated as follows:

$$\begin{aligned}\hat{m}(H,k) &= \sum_{y\in S_H, x_q\in I_H} \lambda(\hat{f}_{qv}>0) - \sum_{y\notin S_H, x_q\in I_H} \lambda(\hat{f}_{qv}>0) \\ &= \sum_{y\in S_H, x_q\in I_H} \lambda([f(\hat{x}_{qv})-f(x_q)]>0) - \sum_{y\notin S_H, x_q\in I_H} \lambda([f(\hat{x}_{qv})-f(x_q)]>0)\end{aligned} \quad (6)$$

where $\lambda(A)\equiv 1$ if a proposition A is true; $\equiv 0$ otherwise, I_H is a set of instances of a schema H, $q = \arg\max_i \hat{f}_{iy}$ and S_H is a set of higher-order schemata of a schema H(for example, if H is 1** then S_H ={1**, 1*0, 1*1, 10*, 11*, 100, 101, 110, 111}). In this case, a string x_i is replaced with the one of the n parasitized strings having best-improved fitness.

Also we can formulate the fitness of a schema H regarding the SCEA from its definition. Let us denote by $f'(H,k)$ the fitness of a schema H after parasitizing process at the generation k. Then

$$f'(H,k) = \frac{\sum_{x\in I_H} f(x) + \sum_{\hat{x}_{iy}\in \hat{I}_H^+} f(\hat{x}_{iy}) - \sum_{x\in \hat{I}_H^-} f(x)}{m(H,k)+\hat{m}(H,k)} \quad (7)$$

where $\hat{I}_H^+$ and $\hat{I}_H^-$ are the index sets of increased and decreased instances of a schema H after the parasitizing process, respectively. Combining the above equations, the schema theorem can be rewritten by

$$m(H,k+1) \geq \frac{f'(H,k)}{\overline{f}(k)} \cdot m'(H,k) \cdot \left[1 - pc \cdot \frac{\delta(H)}{l-1} - p_m \cdot o(H)\right] \quad (8)$$

Since the fitness of a schema H is defined as the average fitness of all strings in the population matched by that schema H, the fitness $f'(H,k)$ of a schema H after parasitized can be approximated by $f'(H,k) \simeq f(H,k)$. Especially, if the number of strings in the host-population $N \gg n$, where n is the number of strings to be parasitized, the above approximation makes sense for the large number of generation sequences [7].

Consequently we obtain an extended schema theorem regarding the SCEA, that is

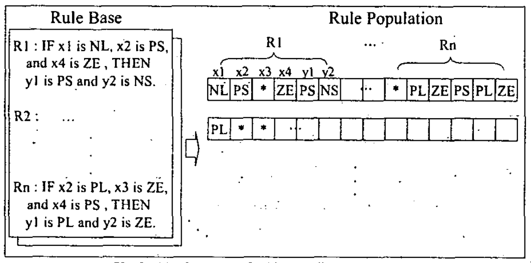
$$\begin{aligned} m(H,k+1) \geq [m(H,k)+\hat{m}(H,k)] \cdot \frac{f(H,k)}{\overline{f}(k)} \\ \cdot \left[1 - p_c \cdot \frac{\delta(H)}{l-1} - p_m \cdot o(H)\right]\end{aligned} \quad (9)$$

Compared with the original Schema Theorem in equation (4), the above equation means that the SCEA allocates much more increasing (decreasing) numbers of trials to the short, low-order, and above- (below-) average schema H than SGA does. Because the parasite-population explores the schema space, global optima could be found more reliably in shorter time than SGA. When the schema containing a solution does not exist in the population, SGA may fail to find global optima. On the other hand, as the useful schema can be found with the parasite-population and it prevails in the host-population via parasitizing process, the SCEA provides much more opportunities to converge on the global optima. We can easily compare the performance of the SCEA with that of SGA by applying them to autonomous mobile robotic system, the objective of which is finding a goal and avoiding static/moving obstacles.

4. CO-EVOLUTION SCHEME IN AMR

4.1. Rule base population

The individual of rule base population consists of a set of rules, so there are sets of rules in the rule population. And a set of rules is made up of ten different rules. If membership functions are partitioned into five terms and there are n preconditions, then the maximum number of IF-THEN fuzzy rules is 5^n. This means that the input space is divided into 5^n. Therefore, unless we use all of the rules, null set problems occur when the given rule base cannot cover the current input states. So we use a don't-care symbol in addition to linguistic terms for a rule chromosome. This don't-care symbol makes the preconditions so inclusive that a small number of rules can cover the whole input space. An example of encoding scheme for several given rules is shown in figure 4.


Rule Base
Rule Population
R1 : IF x1 is NL, x2 is PS, and x4 is ZE , THEN y1 is PS and y2 is NS.
R2 : ...
Rn : IF x2 is PL, x3 is ZE, and x4 is PS , THEN y1 is PL and y2 is ZE.
R1
Rn
x1 x2 x3 x4 y1 y2
NL PS * ZE PS NS
* PL ZE PS PL ZE
PL * *

PL : Positive Large, PS : Positive Small, ZE : Zero,
NS : Negative Small, NL : Negative Large, * : Don't-Care

Fig.4 An example of fuzzy rules encoding scheme

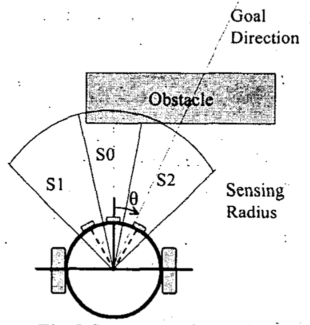


Fig.5 Sensor configuration

In order to ensure the character preservation, we use the Elitism method and a mutation operator only as genetic operators. This selection method is elitist and therefore guarantees a monotonically improving performance.

4.2. Simulation and results

We verify the effectiveness of the proposed algorithm by applying it to optimal path planning of autonomous mobile robot (AMR). The objective of this problem is to find an optimal path when static and moving obstacles exist. For the moving obstacle we assumed that there are two robots with the same fuzzy logic controller (FLC) at the counterpart corner. Each robot's goal position is set to the other robot's starting point and perceives the other robot as an obstacle. A robot has three sensors (S0, S1, S2) covering ±15° to detect the distance to an obstacle. And the direction of its goal (θ) is given, so there are four input variables. For the outputs, FLC gives the directional changes (φ) and speed (v) of AMR.

The input/output variables' ranges are restricted as shown in table 1. And figure 5 shows the AMR's sensor configuration and situations of detecting an obstacle. The size of the AMR is 50mm in diameter.

Table 1. Range of input/output variables

INPUT				OUTPUT	
θ	S0	S1	S2	φ	v
$-180^\circ \sim 180^\circ$	0 ~ 200 mm	0 ~ 200 mm	0 ~ 200 mm	$-90^\circ \sim 90^\circ$	0 ~ 30 mm

And the rule fitness measure is formulated by,

$$fit = (1 - \frac{D_r}{D_G}) \cdot \frac{T_{\min}}{T} \cdot \frac{(N_N - N_n)}{N_N} \qquad (10)$$

where T is consuming time, N_n is the number of null set, $T_{\min}$ is minimum time required to reach the goal, and N_N is maximum number of null set.

In our simulation, the number of rule and schema populations is set for 30 and each individual in the rule population has 16 rules. The linguistic values for θ and φ are NL (Negative Large), NS (Negative Small), ZE (Zero), PS (Positive small), and PL (Positive Large), and the linguistic values for S0, S1, S2, and v are VS (Very Small), MS (Middle Small), ME (Medium), ML (Middle Large), and VL (Very Large). And the mutation probability of rule is 0.2, the crossover and mutation probability of schema populations are set for 0.5 and 0.02, respectively. In each generation, each individual of rule population is evaluated by the robot's action for 300 steps.

We apply SGA and the SCEA to optimal path planning of two AMRs in various static environments. In our simulation, the resulting fitness of the SCEA is increasing faster and has more high value than that of SGA. And both AMRs find their goal positions in relatively short time and avoid obstacles successfully. An example of obtained rules of the AMR after 500 generations is stated in table 2.

Table 2. An example of rule base after 500 generations

R1: IF S1 is ME, and S2 is MS, THEN φ is PS, and v is ME
R2: IF θ is PS, S0 is ML, and S1 is ML, THEN φ is NL, and v is ME
R3: IF θ is NL, and is MS, THEN φ is ZE, and v is ME
R4: IF S0 is VS, and S2 is VS, THEN φ is NS, and v is ME
R5: IF S0 is ME, and S1 is ML, THEN φ is PS, and v is MS
R6: IF θ is ZE, S0 is VL, and S2 is MS, THEN φ is PS, and v is VS
R7: IF θ is PS, and S0 is VS, THEN φ is NS, and v is VL
R8: IF θ is PS, THEN φ is NL, and v is MS

5. CONCLUSIONS

In this paper we proposed the Schema Co-Evolutionary Algorithm to design the rule base of fuzzy logic controller. By applying the proposed method to an optimal path planning problem where moving obstacle exist, the effectiveness of the proposed method was shown. The idea of the Schema Co-Evolutionary Algorithm is based on the schema theorem and building block hypothesis and on the host-parasite co-evolution. The individual of host-population is parasitized by a schema in parasite population. By this process, useful schema generates much more instances in rule populations at the next generation. Also it gives much more chance to find global optima than SGA because the parasite-population searches the schema space.

Acknowledgements

This research was supported by Brain Science and Engineering Research Program sponsored by Korea Ministry of Science and Technology.

6. REFERENCES

[1] John, H. Holland, *Adaptation Natural and Artificial Systems*, Ann Arbor, University of Michigan Press, 1975.
[2] Z. Michalewicz, *Genetic Algorithms+Data Structures Evolution Programs*, Third Edition, Springer-Verlag, 1995.
[3] Melanie Mitchell, *An Introduction to Genetic Algorithm*, A Bradford Book, The MIT Press, 1996.
[4] John, R. Koza, *Genetic Programming: On the Programming of Computers by means of Natural Selection*, A Bradford Book, The MIT Press, 1993.
[5] John, R. Koza, *Genetic Evolution and Co-Evolution of Computer Programs*, Artificial Life II, Addison-Wesley, 1991.
[6] Rechenberg, I, *Cybernetic Solution Path of an Experimental Problem*, Ministry of Aviation, Royal Aircraft Establishment (U.K.), 1965.
[7] Hans-Paul Schwefel, *Evolution and Optimum Seeking*, A Wiley-Interscience Publication, John Wiley & Sons, Inc., 1995.
[8] Fogel, L. J., Owens, A. J., and Walsh, M. J., *Artificial Intelligence Through Simulated Evolution*, John Wiley, Chichester, UK, 1966.
[9] Seth G. Bullock, "Co-Evolutionary Design: Implications for Evolutionary Robotics," *COGS Technical report CSRP* 384, Univ. of Sussex, 1995.
[10] W. Daniel Hillis, "Co-Evolving parasites Improve Simulated Evolution as an Optimization procedure," *Artificial life II*, Vol. X, pp. 313-324, 1991.
[11] Jan Paredis, "Co-Evolutionary Computation," *Artificial life*, Vol. 2, No. 4, pp. 353-375, 1995.
[12] D. W. Lee and K. B. Sim, "Structure Optimization and learning of Neural Networks by Co-Evolution," *Proc. of The Third International Symposium on Artificial Life and Robotics*, Vol. 2, pp. 462-465, 1998.
[13] D. Cliff, G. F. Miller, "Tracking The Red Queen: Measurements of adaptive progress in co-evolution," *COGS Technical Report CSRP* 363, Univ. of Sussex, 1995.
[14] T. Kuo and S. Y. Hwang, "A Genetic Algorithm with Disruptive Selection," *IEEE Trans. On Systems, man and Cybernetics*, vol. 26, No. 2, pp. 299-307, 1996.